



THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

## Polymorphic queries for P2P systems

**Citation for published version:**

Liu, J & Fan, W 2011, 'Polymorphic queries for P2P systems', *Information Systems*, vol. 36, no. 5, pp. 825-842. <https://doi.org/10.1016/j.is.2011.01.001>

**Digital Object Identifier (DOI):**

[10.1016/j.is.2011.01.001](https://doi.org/10.1016/j.is.2011.01.001)

**Link:**

[Link to publication record in Edinburgh Research Explorer](#)

**Document Version:**

Publisher's PDF, also known as Version of record

**Published In:**

Information Systems

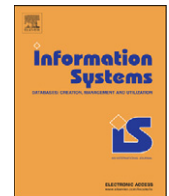
**General rights**

Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**

The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact [openaccess@ed.ac.uk](mailto:openaccess@ed.ac.uk) providing details, and we will remove access to the work immediately and investigate your claim.





# Polymorphic queries for P2P systems

Jie Liu<sup>a,\*</sup>, Wenfei Fan<sup>b</sup>

<sup>a</sup> Department of Computer Science, Ocean University of China, China

<sup>b</sup> School of Informatics, University of Edinburgh, United Kingdom

## ARTICLE INFO

### Article history:

Received 7 August 2010

Received in revised form

3 January 2011

Accepted 11 January 2011

Recommended by: J. Van den Bussche

Available online 18 January 2011

### Keywords:

Polymorphic query

Dependencies Schema mapping

Object identification

## ABSTRACT

When a query is posed on a centralized database, if it refers to attributes that are not defined in the database, the user is warranted to get either an error or an empty set. In contrast, when a query is posed on a peer in a P2P system and refers to attributes not found in the local database, the query should not be simply rejected if the relevant information is available at other peers. This paper proposes a query model for unstructured P2P systems to answer such queries. (a) We introduce a class of polymorphic queries, a revision of conjunctive queries by incorporating type variables to accommodate attributes not defined in the local database. (b) We define the semantics of polymorphic queries in terms of horizontal and vertical object expansions, to find attributes and tuples, respectively, missing from the local database. We show that both expansions can be conducted in a uniform framework. (c) We develop a top-K algorithm to approximately answer polymorphic queries. (d) We also provide a method to merge tuples collected from various peers, based on matching keys specified in polymorphic queries. Our experimental study verifies that polymorphic queries are able to find more sensible information than traditional queries supported by P2P systems, and that these queries can be evaluated efficiently.

© 2011 Elsevier B.V. All rights reserved.

## 1. Introduction

Consider a centralized database  $D$  specified by schema  $S$ . When  $D$  is a centralized database and a query  $Q$  is posed on  $D$ , if  $Q$  refers to attributes not found in  $S$ , then the user is warranted to get either an error or an empty set. This is also the semantics adopted by current query models for P2P systems [1–10]: when  $Q$  is posed on  $D$  residing at a peer  $P$ ,  $Q$  is not allowed to refer to any attributes that are not defined in  $S$ .

However, while the information about an attribute cannot be found in  $D$ , it may be available at other peers in the P2P system. One would expect that P2P systems could do better than centralized database systems. Indeed, as illustrated below, P2P systems may be able to find missing attributes at other peers, and hence, should not simply reject

$Q$ . After all it is to share data that P2P systems are developed in the first place.

**Example 1.1.** Alice is interested in John Denver's albums that received a good rating. She wants to query a P2P system and find information about the price, label and release of those albums. She has only access to peer  $P_0$ . The database at  $P_0$  is specified by schema

`review(album,artist,rating).`

As shown in Fig. 1(a), a review relation collects albums by various artists, and with each album it associates an average rating in the scale [0, 4]. To this end Alice poses an SQL query  $Q_0$  on the database residing at  $P_0$ :

```
select  album, price, label, release
from    review
where   artist = "Denver, J" and rating = "4"
```

Observe that  $Q_0$  refers to attributes price, label and release, which are not defined in the local schema review.

\* Corresponding author.

E-mail addresses: [liujie@ouc.edu.cn](mailto:liujie@ouc.edu.cn) (J. Liu), [wenfei@inf.ed.ac.uk](mailto:wenfei@inf.ed.ac.uk) (W. Fan).

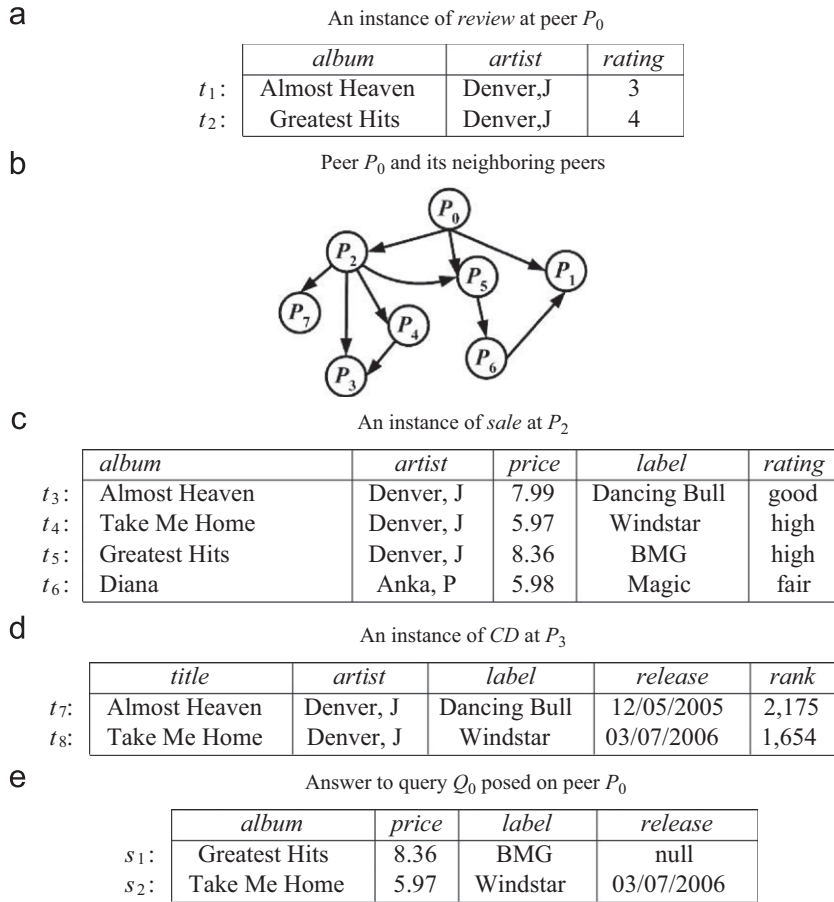


Fig. 1. Example data and answer to query  $Q_0$ .

If  $Q_0$  were posed on a centralized database system, Alice would get either an error or an empty set.

However, while price, label and release are not provided by peer  $P_0$ , they may be available at other peers in the P2P system. As depicted in Fig. 1(b),  $P_0$  has a neighboring peer  $P_2$  with a database of schema *sale*(album, artist, price, label, rating), which in turn has a neighbor  $P_3$  with a database of schema *CD*(title, artist, label, release, rank). Instances of *sale* and *CD* are shown in Figs. 1(c) and (d), respectively.

Provided these, the system has got enough information to answer  $Q_0$ . (1) For the album “Greatest Hits” found at  $P_0$ , we can find its price and label from  $P_2$ . That is, we can “horizontally” expand the object by including missing attributes found at other peers. In addition, (2) from  $P_2$  an album “Take Me Home” is found (when ‘high’ indicates a good rating), which is missing from  $P_0$ . The answer to  $Q_0$  can be “vertically” expanded by including the album, which is further expanded at  $P_3$  by adding release. Taken together, the answer to  $Q_0$  contains tuples shown in Fig. 1(e). □

Several query models have been put forward for unstructured P2P systems, based on, e.g., schema mapping and certain query answering [1–3], information retrieval [4], mapping (concordance) tables [5], approximate query processing [7,8], dynamic construction of group schemas [9,10],

and query expansion via synonym rules [11] (see [12] for a recent survey). However, we are not aware of any P2P query models that allow queries to explicitly retrieve attributes not defined in a local schema, such as the query  $Q_0$  given above. This highlights the need for a new P2P query model to support such queries.

**Contributions.** To explore the data sharing nature of P2P systems, we propose a query model for unstructured, schema-heterogeneous P2P systems. The model consists of (1) a revision of conjunctive (SPC) queries that may refer to attributes not defined in the local schema, (2) the semantics of the queries in terms of object expansions, (3) an efficient top-K algorithm for approximately answering the queries, and (4) a method for merging tuples from various peers that represent the same real-world object.

(1) We introduce a class of queries for P2P systems, referred to as *polymorphic queries*. Polymorphic queries extend SPC queries by supporting: (a) *type variables* to specify, explicitly or implicitly, attributes that are not defined in a local schema, and (b) *matching keys* to guide how tuples retrieved from various peers are merged. For example, query  $Q_0$  can be expressed as a polymorphic query.

Polymorphic queries are based on the notion of extensible records, which have proved extremely useful in functional programming (see, e.g., [13]). An extensible

record carries *unknown* fields with type variables, in addition to a set of known fields with fixed types. It allows one to build an object *incrementally* by adding a finite number of fields and instantiating their type variables accordingly. Along the same lines, we use type variables to cope with attributes found at other peers that are “unknown” to the local peer.

(2) We define the semantics of polymorphic queries in terms of two forms of expansions: (a) *horizontal expansion*, to augment an object by incorporating additional attributes of the object found at various peers in the P2P system, and (b) *vertical expansion*, to enrich query answer by including tuples missing from the local peer but found at other peers in the system.

Referring to query  $Q_0$ , horizontal and vertical expansions yield tuples  $s_1$  and  $s_2$  of Fig. 1(e) in the answer to  $Q_0$ , respectively. Most P2P query models typically support vertical expansion only.

We provide a conceptual evaluation strategy for polymorphic queries, conducting horizontal and vertical expansions in a uniform framework based on a notion of *contextual foreign keys* (CFKs). CFKs extend foreign keys that reference primary keys, by incorporating patterns of semantically related data values. They specify correspondences between *attributes* and between *values* across different peers. They can express lexical semantic relations as found in, e.g., WordNet (see <http://en.wikipedia.org/wiki/WordNet>). Instead of assuming the existence of schema mapping [2,3] or mapping tables [5], we show that CFKs between neighboring peers suffice to rewrite queries for vertical expansion and to collate tuples for horizontal expansion.

(3) To reduce the communication cost of the conceptual evaluation strategy, we develop a top-K algorithm to approximately answer polymorphic queries. The algorithm decides whether a query and relevant objects should be forwarded from one peer to another, based on a *quality model*. The model takes into account of the local data at the peer and the statistics of the data at its neighboring peers. Given a query  $Q$  and predefined numbers  $K$  and  $m$ , the algorithm returns  $K$  top tuples in the answer to  $Q$ , with a performance guarantee: at each peer it forwards at most  $K$  tuples to at most  $m$  neighbors. In addition, we present optimization methods to further reduce network traffic.

(4) Tuples collected from various peers may represent the same real-world object. We provide a method to merge such tuples, an issue that has not been well explored for P2P systems [14]. Our approach is based on a notion of matching keys, which may be optionally specified in a polymorphic query. To identify tuples from different sources, a matching key specifies what attributes to compare and how to compare them, in terms of equality or similarity operators.

(5) We experimentally verify that polymorphic queries and their evaluation techniques are capable of finding far more relevant information than the traditional approach based on schema mapping (e.g., [2]), without substantial degradation in performance. Indeed, the conceptual evaluation strategy constantly retrieves 5 times more results than the mapping-based approach, and the top-K

algorithm finds 1.42 times more than the traditional approach (when  $K \geq 40$  and  $m = 3$ ). When top  $K$  tuples are concerned, the top-K algorithm finds up to 84.78% of the results of the conceptual strategy, and 253% more than that of the mapping approach, with significantly less network traffic. Moreover, both the conceptual strategy and the top-K algorithm scale well with the number of peers and the size of data. We also find that our tuple merging method is effective: it constantly identifies over 19% of tuples returned by the traditional approach that refer to the same object.

**Organization.** We discuss related work in Section 2. Polymorphic queries are introduced in Section 3, followed by CFKs in Section 4. The semantics of polymorphic queries is defined in Section 5 by giving the conceptual evaluation strategy based on object expansions. The top-K algorithm is developed in Section 6, followed by a method for tuple merging in Section 7. The experimental study is presented in Section 8, followed by conclusions in Section 9.

## 2. Related work

Several query models have been studied for unstructured P2P systems (e.g., [1–5,9–11]). Piazza [2] interprets P2P queries based on schema mappings (query rewriting) and certain query answering. A variation was proposed in [3] in terms of epistemic FO (First-Order logic). This approach is effective when neighboring peers do not have radically different schemas. However, successive rewritings often reduce information that a query is to retrieve, when e.g., attributes at one peer do not find a match at its neighbors. To tackle this problem, automated construction of group schemas was studied in [9,10]. PeerDB [4] is based on information retrieval techniques. Heptox [1] uses rules to translate queries. These models support vertical expansion, but not horizontal expansion. They do not expand objects by adding relevant *attributes* retrieved from other peers.

Closer to this work is Hyperion, based on mapping tables [5]. A mapping table maintains value and name correspondences between data in neighboring peers, which is similar to CFKs. Hyperion evaluates a query  $Q$  by traversing peers, translating  $Q$  to a set of queries based on mapping tables, and collecting relevant objects found by those translated queries. It simply puts these objects together via outer union, but considers neither extending existing objects with additional attributes nor conflict resolution.

A notion of query expansion has been explored for P2P queries [6]. It is to enhance queries with vague or surrounding concepts of pre-defined keywords, when users are unable to identify precise keywords. This is quite different from object expansion: query expansion aims to find more relevant results of fixed keywords, and is developed mostly for keyword queries [15,16]; in contrast, object expansion is to enrich objects with new relevant attributes, for SPC queries.

To the best of our knowledge, no previous P2P models allow queries to explicitly refer to attributes that are not defined at the local peer, such as  $Q_0$  of Example 1.1.

Quality models have been proposed in, e.g., [7,8] to select peers, based on certain metrics of information completeness. Top-K algorithms have also been studied for various applications (see [17] for a survey). The quality model proposed in this work differs from prior models in that it takes into account not only new information retrieved via vertical expansion, but also the amount of new information associated with attributes added via horizontal expansion.

Conflict resolution has been studied for data integration (e.g., [18–20]) and uncertain data (e.g., [21,22]). There has also been a host of work on record matching (see [23] for a survey). As observed in [14], however, few P2P query models deal with conflicts. This work is among the first efforts to explore these issues for P2P queries.

There has also been work on polymorphic type inference for relational algebra [24]. The focus is to determine on what schema a query  $Q$  is well defined, and to infer the “principle” (most generic) type for  $Q$ . This is studied for centralized systems. In contrast, given a query  $Q$  that is *not* well defined at the local peer in the standard semantics, this work studies how to evaluate the query based on object expansion. This also involves object merge and conflict handling, which are not encountered in polymorphic type inference.

### 3. Polymorphic queries

We next present the syntax of polymorphic queries. To simplify the discussion we define polymorphic queries as an extension of SPC queries, and defer the study of more general polymorphic queries to future work.

**SPC queries.** An SPC query [25] is defined on a relational schema  $\mathcal{R}$  in terms of the *selection* ( $\sigma$ ), *projection* ( $\pi$ ) and *Cartesian product* ( $\times$ ) operators. It is of the form:

$$\pi_L(\sigma_F(E_c)) \quad \text{where } E_c = R_1 \times \dots \times R_n.$$

Here (a) for each  $j \in [1, n]$ , we assume w.l.o.g. that  $R_j$  is a relation atom in  $\mathcal{R}$  such that the attributes in  $R_j$  and  $R_l$  are disjoint if  $j \neq l$ ; (b)  $F$  is a predicate built from equality atoms such as  $A=B$  and  $A = 'a'$  for a constant  $a$  in the domain of  $A$ , by closing under conjunction (and) and disjunction (or); and (c) let  $Q_c$  denote  $\sigma_F(E_c)$ , then in the projection  $\pi_L(Q_c)$ ,  $L$  is a list of attributes *appearing* in  $Q_c$ .

Note that we allow disjunction in  $F$  and hence, support certain SPCU queries defined with union.

**Polymorphic queries.** We define an extension of SPC, referred to as *polymorphic queries* and denoted by  $\text{SPC}^*$ , by supporting (1) a *polymorphic projection* operator  $\Pi$  with type variables, and (2) an optional list MK of matching keys:

$$\Pi_L(Q_c) \text{ group\_by MK} \quad \text{where } L = (L_1; L_2; \alpha).$$

Here (a)  $Q_c = \sigma_F(E_c)$  is the same as above, (b)  $L_1$  is a list of attributes appearing in  $Q_c$ , but in contrast, (c)  $L_2$  is a list of attributes *not* appearing in  $Q_c$ , and (d)  $\alpha$  is an optional variable, which, if present, is to be instantiated with a list of attributes appearing in neither  $L_1$  nor  $L_2$ . Intuitively,

- $L_2$  denotes attributes that the user *explicitly* wants to find from a P2P system, although they are not defined

in the local schema; while the labels of these attributes are known, their *types* are *unknown* and are represented by type variables; and

- $\alpha$  indicates that other relevant attributes are also demanded, if any. The user needs to know neither the *labels* nor the *types* of these attributes.

We refer to  $L_1$  attributes as *local attributes*, and  $L_2$  as *explicit attributes*. We refer to attributes that instantiate  $\alpha$  as *implicit attributes*. Note that both  $L_2$  and  $\alpha$  carry type variables, along the same lines as extensible records [13].

We denote by  $\text{sch}(Q)$  the *output schema* of an  $\text{SPC}^*$  query  $Q$ , i.e., the schema of the answer to  $Q$  in a P2P system. Here  $\text{sch}(Q)$  consists of local, explicit and implicit attributes.

**Example 3.1.** The query  $Q_0$  described in Example 1.1 can be expressed as an  $\text{SPC}^*$  query  $Q = \Pi_{(L_1; L_2)}(\sigma_F(E_c))$ , where (a)  $F$  is a conjunction of equality atoms artist = “Denver, J” and rating = “4”; (b)  $E_c$  is the review relation schema at the local peer  $P_0$ ; (c)  $L_1 = [\text{album}]$  and  $L_2 = [\text{price}, \text{label}, \text{release}]$ , which are the attributes appearing in  $\text{sch}(Q_0)$ .

As another example, Alice may want to retrieve other relevant attributes, although she does not know the labels of those attributes. To this end she may write query  $Q_1$  by adding  $\alpha$  to  $Q_0$ :  $\Pi_{(L_1; L_2; \alpha)}(\sigma_F(E_c))$ . When  $Q_1$  is evaluated in the P2P system described in Example 1.1,  $\alpha$  will be instantiated with attributes found in the system, including but not limited to rank from  $P_3$ . The output schema  $\text{sch}(Q_1)$  consists of all these attributes and those in  $L_1$  and  $L_2$ , with type variables instantiated with the corresponding domains.

Extending the SQL syntax,  $Q_1$  can be written as:

```
select*  album; price, label, release;  X
from    review
where   artist = "Denver, J" and rating = "4"
```

Here  $X$  indicates the variable  $\alpha$  in the  $\text{SPC}^*$  query.  $\square$

**Remark.** (1) Just like SPC queries, when writing an  $\text{SPC}^*$  query, a user does not need to know anything *beyond* the local schema. She may declaratively request other attributes of interest, no matter whether she knows their labels or not, as if they were defined in the local schema. As will be seen shortly, it is the polymorphic query model that automatically retrieves those “external” attributes across the entire system. This provides the user with the flexibility and expressive power to share data in the P2P system.

(2) When an SPC query  $Q'$  is posed on a centralized database  $D$ , the output schema of  $Q'$  is uniquely determined by  $Q'$  and the schema of  $D$ . In contrast, we cannot *statically determine*  $\text{sch}(Q)$  of an  $\text{SPC}^*$  query  $Q$  in a P2P system based on  $Q$  and the local schema alone. Referring to Example 1.1, for instance, one cannot determine the types of price, label, and release based on  $Q_0$  and the schema review at compile time. The output schema is “open-ended” and is *incrementally completed* when the query is evaluated by traversing the linked peers, along the same lines as extensible records.



(3)  $\text{SPC}$  queries are a special case of  $\text{SPC}^*$  queries: when  $L_2$  is empty, and when  $\alpha$  and  $\text{MK}$  are absent.

(4) To simplify the discussion we have assumed that distinct type variables for attributes in  $L_2$  are automatically generated. Following extensible records of functional programming [13], this model can be extended by allowing multiple attributes to share the same type variable. For instance, if *firstname* and *lastname* are attributes in  $L_2$ , we may require *firstname* and *lastname* to carry the same type variable  $\tau$ , i.e., the two attributes will bear the same type no matter how  $\tau$  is instantiated. As will be seen in Section 7, such typing constraints can be enforced in the tuple merging phase, which resolves typing conflicts for tuples collected from different peers.

**Matching keys.** In an  $\text{SPC}^*$  query  $Q$ , if  $\text{MK}$  is specified, it is of the form  $\phi_1, \dots, \phi_m$ . Each  $\phi_j$  ( $j \in [1, m]$ ) is a *matching key* specified by a set of attribute-operator pairs:  $((A_1, \text{op}_1), \dots, (A_l, \text{op}_l))$ . Here for each  $i \in [1, l]$ ,  $A_i$  is an attribute in  $L_1$  or  $L_2$ , and  $\text{op}_i$  is either a *similarity* operator ' $\approx$ ' or an equality ' $=$ '.

A matching key expresses a matching rule of [26]. We assume a set  $\Theta$  of similarity metrics such as  $q$ -grams, Jaro distance or edit distance [23]. For each  $\approx$  in  $\Theta$  and for values  $x$  and  $y$ ,  $x \approx y$  yields true iff  $x$  and  $y$  are “close” enough in the similarity metric  $\approx$  w.r.t. a predefined threshold.

Tuples  $s_1$  and  $s_2$  satisfy  $\phi_j$  if  $s_1[A_i] \text{op}_i s_2[A_i]$  for all  $i \in [1, l]$ , i.e., the attributes of  $\phi_j$  in  $s_1$  and  $s_2$  pairwise “match” w.r.t. the corresponding similarity operators. Intuitively, if  $s_1$  and  $s_2$  satisfy  $\phi_j$  then they represent the same object.

**Example 3.2.** Consider a tuple  $s_3$ : (*album* = “The Greatest Hits”, *price* = 9.99, *label* = “BMG”, *release* = 01/07/2002), found at, e.g.,  $P_4$  (Fig. 1(b)). Then  $s_3$  and  $s_1$  of Fig. 1(e) represent the same album. However,  $s_1 \neq s_3$  if one attempts to compare them pairwise w.r.t. all attributes in  $\text{sch}(Q_0)$ .

Suppose that a matching key  $\phi$  for query  $Q_0$  is specified:  $((\text{album}, \approx), (\text{label}, =))$ , where  $\approx$  is a similarity metric such that “The Greatest Hits”  $\approx$  “Greatest Hits”. Then  $s_1$  and  $s_3$  satisfy  $\phi$  and can be identified, although some of their attributes, e.g., *price*, are radically different.  $\square$

Intuitively, matching keys are an extension of traditional relational keys by incorporating similarity operators to accommodate errors or different representations in tuple matching. They allow us to group (cluster) tuples in the answer to  $Q$  by attributes in the keys, such that tuples in the same group are identified to represent the same real-world object.

#### 4. Contextual foreign keys

To give the semantics of  $\text{SPC}^*$  queries, we first define contextual foreign keys, an extension of foreign keys.

**Foreign keys (FKs).** Recall [25] that a foreign key  $\text{fk}$  from relation  $R_1$  to  $R_2$  is of the form  $R_1[X] \subseteq R_2[Y]$ , where  $X, Y$  are attribute lists in  $R_1, R_2$ , respectively, and  $Y$  is a key of  $R_2$ .

Note that  $\text{fk}$  specifies a pair of constraints: (a) a key  $Y$  for  $R_2$ , and (b) an inclusion dependency from  $R_1$  to  $R_2$ .

Given instances  $(D_1, D_2)$  of  $(R_1, R_2)$ ,  $\text{fk}$  asserts that  $Y$  is a key of  $D_2$  and furthermore, for any tuple  $t_1$  of  $D_1$ , there is a tuple  $t_2$  of  $D_2$  such that  $t_1[X] = t_2[Y]$ , i.e.,  $t_1[X]$  references the tuple  $t_2$  identified by  $t_2[Y]$ .

One may want to use FKs to specify schema matching and data concordance across peers. However, FKs (even inclusion dependencies) are not powerful enough, as illustrated below.

**Example 4.1.** Recall relations *review*, *sale* and *CD* from Example 1.1. Suppose that (*album*, *artist*) is the primary key of *review* and *sale*, and (*title*, *artist*) is the primary key of *CD*. One might want to specify FKs from *review* to *sale*, and from *sale* to *CD* as follows:

<i>review</i> ( <i>album</i> , <i>artist</i> )	$\subseteq$	<i>sale</i> ( <i>album</i> , <i>artist</i> )
<i>sale</i> ( <i>album</i> , <i>artist</i> )	$\subseteq$	<i>CD</i> ( <i>title</i> , <i>artist</i> )

These FKs do not make sense if (a) the *review* relation contains information about all albums while *sale* contains only albums that received a rating above “poor”; and (b) the *CD* relation contains only albums from either *Windstar* or *DancingBull*. In light of this, the correspondences from *review* to *sale* and from *sale* to *CD* cannot be expressed as traditional FKs or even as more general inclusion dependencies.  $\square$

**Contextual foreign key (CFKs).** The limitations of traditional FKs motivate us to propose CFKs.

A CFK  $\varphi$  from relation  $R_1$  to  $R_2$  is of the form

$$(R_1[X] \subseteq R_2[Y], t_p[X_p, Y_p]),$$

where (a)  $X, X_p$  (resp.  $Y, Y_p$ ) are two disjoint lists of attributes in  $R_1$  (resp.  $R_2$ ); (b)  $R_1[X] \subseteq R_2[Y]$  is an FK, where  $X$  (resp.  $Y$ ) is a *primary key* of  $R_1$  (resp.  $R_2$ ); (c)  $t_p$  is the *pattern tuple* of  $\varphi$  with attributes in  $X_p$  and  $Y_p$ , such that for each attribute  $B$  in  $X_p$  (or  $Y_p$ ),  $t_p[B]$  is a constant in  $B$ 's domain. We refer to  $t_p[X_p]$  (resp.  $t_p[Y_p]$ ) as the  $X_p$  (resp.  $Y_p$ ) *pattern* of  $\varphi$ .

Instances  $(D_1, D_2)$  of  $(R_1, R_2)$  satisfy  $\varphi$  if  $X$  (resp.  $Y$ ) is the primary key of  $D_1$  (resp.  $D_2$ ) and moreover, for each tuple  $t_1$  in  $D_1$ , if  $t_1[X_p] = t_p[X_p]$ , then there exists a tuple  $t_2$  in  $D_2$  such that  $t_1[X] = t_2[Y]$  and  $t_2[Y_p] = t_p[Y_p]$ .

Intuitively, the  $X_p$  *pattern* of  $\varphi$  identifies a subset of  $D_1$  that matches  $t_p[X_p]$ , and the traditional FK  $R_1[X] \subseteq R_2[Y]$  is enforced on this subset rather than on the entire  $D_1$ . Further, for each tuple  $t_2$  in  $D_2$  that is referenced by  $t_2[Y]$  (i.e.,  $t_1[X]$ ), the  $Y_p$  *pattern* is enforced, i.e.,  $t_2[Y_p] = t_p[Y_p]$ .

**Example 4.2.** The constraints described in Example 4.1 can be expressed as CFKs as follows:

$\varphi_1$ :	( <i>review</i> ( <i>album</i> , <i>artist</i> ) $\subseteq$ <i>sale</i> ( <i>album</i> , <i>artist</i> ),
	( <i>review</i> ( <i>rating</i> ) = “4”, <i>sale</i> ( <i>rating</i> ) = “high”))
$\varphi_2$ :	( <i>review</i> ( <i>album</i> , <i>artist</i> ) $\subseteq$ <i>sale</i> ( <i>album</i> , <i>artist</i> ),
	( <i>review</i> ( <i>rating</i> ) = “3”, <i>sale</i> ( <i>rating</i> ) = “good”))
$\varphi_3$ :	( <i>sale</i> ( <i>album</i> , <i>artist</i> ) $\subseteq$ <i>CD</i> ( <i>title</i> , <i>artist</i> ), ( <i>sale</i> ( <i>label</i> ) = “Windstar”))
$\varphi_4$ :	( <i>sale</i> ( <i>album</i> , <i>artist</i> ) $\subseteq$ <i>CD</i> ( <i>title</i> , <i>artist</i> ), ( <i>sale</i> ( <i>label</i> ) = “DancingBull”))

Here CFK  $\varphi_1$  asserts that for each tuple  $t_i$  ( $i \in [1, 2]$ ) in the *review* relation, if  $t_i[\text{rating}] = “4”$ , then there must be

a tuple  $t_j$  in the sale relation such that  $t_i$  and  $t_j$  agree on their (album, artist) attributes and moreover,  $t_j[\text{rating}] = \text{“high”}$ ; similarly for  $\varphi_2$ . These two CFKs ensure that review tuples can be mapped to sale tuples if and only if their ratings are above 2, and moreover, that a rating of “4” (resp. “3”) in review corresponds to “high” (resp. “good”) in sale.

The CFKs  $\varphi_3$  and  $\varphi_4$  assure that sale tuples can find a match in the CD relation only for those albums from either Windstar or DancingBull. Note that no  $Y_p$  patterns are specified for CD in  $\varphi_3$  and  $\varphi_4$ .  $\square$

**Remark.** Observe the following. (1) Traditional FKs are a special case of CFKs with empty  $X_p$  and  $Y_p$ , when  $X$  (resp.  $Y$ ) is the primary key of  $R_1$  (resp.  $R_2$ ). (2) CFKs are a variation of conditional inclusion dependencies (CINDs) studied in [27]. A CFK specifies three constraints: (a)  $X$  is the primary key of  $R_1$ , (b)  $Y$  is the primary key of  $R_2$ , and (c) an inclusion dependency from  $R_1$  to  $R_2$  with a pattern. In contrast, CINDs specify (c) alone without requiring (a) or (b). (3) CFKs can express lexical semantic relations (e.g., WordNet).

We shall give the semantics of polymorphic queries in terms of CFKs in the next section. More specifically, we use CFKs to collate information about the same object for horizontal expansion, and to rewrite queries for vertical expansion. This is a departure from previous P2P models based on schema mapping [3,2,8], as illustrated by the example below.

**Example 4.3.** Consider schemas review and sale given in Example 1.1, for databases at peers  $P_0$  and  $P_2$ , respectively. Suppose that one wants to specify “peer mapping” [2] from  $P_0$  to  $P_2$  as schema mapping  $Q_{(0,2)}$ , i.e.,  $Q_{(0,2)}$  is a query from instances of review to instances of sale. By treating review as a “mediated schema” and sale as a “data source”,  $Q_{(0,2)}$  is a *local-as-view* (LAV) mapping [2,28]. One can see the following. (1) The instance  $D_2$  of sale at peer  $P_2$  cannot be an exact view [28] of the instance  $D_0$  of review at peer  $P_0$ , no matter what query  $Q_{(0,2)}$  is considered. Indeed, as we can see from Figs. 1(a) and (c), tuples  $t_4$  and  $t_6$  of  $D_2$  cannot find a match in  $D_0$ , and moreover, although  $t_3$  and  $t_5$  of  $D_2$  have a match in  $D_0$ , their price and label attributes are not mapped from tuples in  $D_0$ . (2) One might want to consider a combination of *global-as-view* (GAV) and LAV as suggested in [2], i.e., a pair  $Q_{(0,2)}$  and  $Q_{(2,0)}$  of queries such that  $Q_{(0,2)}(D_0) = Q_{(2,0)}(D_2)$ . However, it is nontrivial to find such mappings. Indeed, schema mapping is often derived from inclusion dependencies across peers [29]. Deriving schema mapping from dependencies is itself a computationally intractable problem [30]. (3) Even when schema mappings from review to sale are in place, they do not tell us how to answer query  $Q_0$  posed on peer  $P_0$ . Indeed, the mappings do not specify how tuple  $t_2$  of  $D_0$  is related to tuples of  $D_2$ , such that  $t_2$  can be horizontally expanded by including attributes price and label. Neither query rewriting nor query unfolding helps here.

There has also been recent work on data exchange, *a.k.a.* schema mapping (see [31] for a recent survey). Data exchange is often specified in terms of constraints, such

**Table 1**

A summary of notations.

Notation	Name	Definition
SPC*	Polymorphic queries	$\Pi_L(Q_c)$ group_by MK
	Local attributes $L_1$ ; explicit and implicit attributes $L_2$ and $\alpha$ with type variables	
MK	Matching keys	$((A_1, \approx_1), \dots, (A_l, \approx_l))$
	Rules for identifying tuples by comparing attributes $A_i$ via similarity operators $\approx_i$	
CFK	Contextual foreign keys	$(R_1[X] \subseteq R_2[Y], t_p[X_p, Y_p])$
	Foreign keys with patterns $t_p[X_p, Y_p]$ , for horizontal and vertical expansions	

as tuple generating dependencies (TGDs). However, data exchange aims to materialize a target instance using data from a data source, e.g., to generate a sale instance from the instance  $D_0$  of review. It is not for answering queries posed on  $D_0$  with data in  $D_2$ . Furthermore, while TGDs are more expressive than CFKs, they do not tell us whether a tuple in  $D_0$  and another in  $D_2$  refer to the same entity, as opposed to CFKs. That is, the increased expressive power of TGDs does not help when it comes to horizontal expansion, not to mention the extra complexity when reasoning about TGDs.  $\square$

For the ease of reference we summarize various notations in Table 1.

## 5. The semantics of SPC\* queries

We now present the semantics of polymorphic queries in a P2P system based on CFKs. To focus on the main idea of horizontal and vertical expansions, we first give a conceptual evaluation strategy that may not be efficient, and defer the presentation of optimization techniques to Section 5.5.

### 5.1. A conceptual query evaluation strategy

Consider an unstructured P2P system  $\mathcal{P} = (P_0, \dots, P_n)$ , where  $P_j$  is a peer. For each  $j \in [0, n]$ , the peer  $P_j$  is specified by:

- the relational schema  $S_j$  of its local database  $D_j$ , such that on each relation  $R$  in  $S_j$ , a primary key is defined; and
- for those  $i \in [0, n]$  such that  $P_i$  is a neighboring peer of  $P_j$ , a set  $\Sigma_{(j,i)}$  of CFKs from relations of  $S_j$  to relations of  $S_i$ , stored at peer  $P_i$ .

**Remark.** We do *not* require that  $(D_j, D_i)$  satisfy  $\Sigma_{(j,i)}$ . Indeed, we simply use the CFKs to specify how *attributes* and *data values* across different peers are mapped to each other, at the schema level. These CFKs can be either explicitly specified or automatically discovered when a new peer joins the system. The maintenance cost for the CFKs is no larger than its counterparts for schema

mapping or mapping tables. As shown in Example 4.3, it is less demanding to assume CFKs  $\Sigma_{(j,i)}$  rather than schema mapping.

To simplify the exposition, we assume that all CFKs defined on a relation  $R$  of  $S_j$  in  $\Sigma_{(j,i)}$  reference a unique  $R'$  in  $S_i$ , i.e., each  $R$  is mapped to at most one relation in  $S_i$  via CFKs, as often assumed in schema mapping. Note that there may exist multiple CFKs from  $R$  to  $R'$  in  $\Sigma_{(j,i)}$ , and there may be CFKs from  $R$  to distinct  $R'$ s in  $\Sigma_{(j,l)}$  for  $l \neq i$ .

**Evaluation.** Consider an  $\text{SPC}^*$  query  $Q = \Pi_L(Q_c)$  group\_by MK, where  $L = (L_1; L_2; \alpha)$ . Suppose that  $Q$  is posed on peer  $P_0$ , referred to as the local peer. Then as depicted in Fig. 2,  $Q$  is evaluated in the P2P system  $\mathcal{P}$  as follows.

**Initial answer.** Upon receiving  $Q$ , local peer  $P_0$  generates a set  $\text{ans}_0$  of objects as follows. (a) It first rewrites  $Q$  into a normal  $\text{SPC}$  query  $Q'$  on its local database  $D_0$ . (b) It then extracts a set  $\text{ans}_0 = Q'(D_0)$  of tuples. (c) Finally, it forwards  $Q$  and  $\text{ans}_0$  to all of its neighboring peers for expansion.

**Expand and forward.** When peer  $P_i$  receives query  $Q_j$  and a set  $\text{ans}_j$  of tuples from peer  $P_j$ , where  $Q_j$  is a rewriting of  $Q$  and is defined on the database  $D_j$ ,  $P_i$  expands  $\text{ans}_j$  horizontally and vertically as follows:

- **horizontal:** leveraging CFKs in  $\Sigma_{(j,i)}$ ,  $P_i$  extends tuples in  $\text{ans}_j$  by adding relevant attributes available at  $P_i$ ;
- **vertical:** it extracts new tuples from its database  $D_i$ . More specifically, using  $\Sigma_{(j,i)}$ , it first rewrites  $Q_j$  into query  $Q_i$  that is defined on  $D_i$ . It then executes  $Q_i$  on  $D_i$ , and expands  $\text{ans}_j$  with new tuples in  $Q_i(D_i)$ .

As shown in Fig. 2,  $P_i$  generates two sets of tuples: (a)  $\text{new}_i$  consisting of tuples not in  $\text{ans}_j$  that are added by vertical expansion, and those tuples in  $\text{ans}_j$  extended with new attributes by horizontal expansion; and (b)  $\text{ans}_i = \text{new}_i \cup \text{ans}_j$ .

Peer  $P_i$  sends  $\text{new}_i$  back to the local peer  $P_0$  as part of the answer to  $Q$  in the P2P system. Meanwhile it forwards  $Q_i$  and  $\text{ans}_i$  to its neighboring peers for further expansions, which forms “forward and expand” paths.

As will be seen in Section 5.5, both  $\text{new}_i$  and  $\text{ans}_i$  can be significantly reduced based on our optimization techniques. That is, the “expand and forward” step does not necessarily incur heavy network traffic.

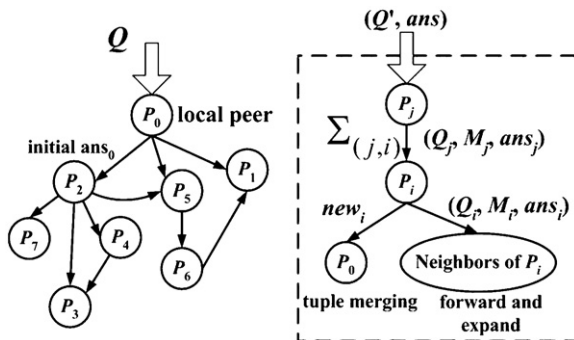


Fig. 2. Polymorphic query evaluation.

**Tuple merging.** The expansion process proceeds until no more attributes or tuples are sent to  $P_0$ . Then the local peer  $P_0$  identifies and merges tuples representing the same real-world object, and handles conflicts based on matching keys. At this stage it instantiates the type variables of  $L_2$  attributes as well as implicit attributes (if  $\alpha$  is specified). Here  $L_2$  attributes may be null, and  $\alpha$  may be instantiated to an empty list, as shown in Fig. 1(e). The result is returned as the answer to query  $Q$ .

**Remark.** The complete answer to  $Q$  in the P2P system is the inflational fixpoint of its expansions, which is guaranteed to be reached when the system is relatively “static” (see e.g., [25] for discussions of fixpoint).

Like all other P2P query models in use, one may adopt “time to live (TTL)” (e.g., [32]): the merging step starts when TTL expires. That is, we can use TTL to compute approximate query answers and strike a balance between the complete answer set and the overhead.

**Example 5.1.** Recall  $\text{SPC}^*$  query  $Q_0$  given in Example 1.1, the P2P system shown in Fig. 1, the CFKs of Example 4.2, and the matching keys of Example 3.2. To give an overview of the evaluation strategy, we show how the answers to  $Q_0$  in the system are generated in various stages. We shall present detailed algorithms and examples in the rest of the section.

When  $Q_0$  is posed on the local peer  $P_0$ , the initial answer consists of  $s_0$ , which is extracted from  $t_2$  of relation review:

	album	artist	rating
$s_0$ :	Greatest Hits	Denver, J	4

The initial answer is forwarded to peer  $P_2$  and expanded there:

	album	artist	price	label	rating
$s_1$ :	Greatest Hits	Denver, J	8.36	BMG	high
$s_2$ :	Take Me Home	Denver, J	5.97	Windstar	high

Here  $s_1$  is horizontally expanded by including attributes price and label extracted from  $t_5$  of relation sale, and  $s_2$  is added by vertical expansion with tuple  $t_4$  of sale. The answer set is then forwarded to peer  $P_3$  and expanded there as follows:

	album	artist	price	label	rating	release
$s_1$ :	Greatest Hits	Denver, J	8.36	BMG	high	null
$s_2$ :	Take Me Home	Denver, J	5.97	Windstar	high	03/07/2006

Here  $s_2$  is horizontally expanded by adding attribute release of  $t_8$ . Observe that  $s_1$  is not expanded at  $P_3$  since it does not find a matching tuple in relation CD.

Suppose that peer  $P_4$  has a tuple  $s_3 = (\text{album} = \text{“The Greatest Hits”}, \text{price} = 9.99, \text{label} = \text{“BMG”}, \text{release} = 01/07/2002)$  (see Example 3.2). Then  $s_3$  is added to the answer set via vertical expansion at  $P_4$ .



Taken together, the answer set to  $Q_0$  becomes

	album	artist	price	label	rating	release
$s_1$ :	Greatest Hits	Denver, J	8.36	BMG	high	null
$s_2$ :	Take Me Home	Denver, J	5.97	Windstar	high	03/07/2006
$s_3$ :	The Greatest Hits	Denver, J	9.99	BMG	null	01/07/2002

Finally, the tuples are merged by using matching keys at the local peer:

	album	price	label	release
$s_1$ :	{Greatest Hits, The Greatest Hits}	{8.36, 9.99}	BMG	01/07/2002
$s_2$ :	Take Me Home	5.97	Windstar	03/07/2006

This is final answer set to  $Q_0$  in the P2P system.  $\square$

In the rest of the section we provide the details of initial answer generation, horizontal and vertical expansions, as well as optimization techniques for reducing communication cost. We defer the discussion of tuple merging to Section 7.

## 5.2. Generating initial answer

The local peer  $P_0$  generates an initial answer  $ans_0$  to query  $Q$  by extracting data from its local database  $D_0$ . As shown in Fig. 2, the set  $ans_0$  of tuples is forwarded to and expanded at other peers in the P2P system.

### Procedure Normalize

*Input:* An SPC\* query  $Q = \Pi_{L_1; L_2; \alpha}(\sigma_F(R_1 \times \dots \times R_l))$ .  
*Output:* A set  $ans_0$  of tuples, and a renaming table  $M_0$ .

1.  $M_0 := \emptyset$ ;  $L := L_1$ ;
2. **for** each relation  $R$  in  $R_1, \dots, R_l$  with primary key  $Z$  **do**
3.      $M_0 := M_0 \cup \{(R, Z) \mapsto (R, Z)\}$ ;
4.      $L := L \cup \text{attr}(R)$ ;
5.  $Q' := \pi_L(\sigma_F(R_1 \times \dots \times R_l))$ ;
6.  $ans_0 := Q'(D_0)$ ;
7. **return** ( $ans_0, M_0$ );

Fig. 3. Algorithm Normalize.

The SPC\* query  $Q$  may be defined with attributes not in  $D_0$ , and thus cannot be directly executed against  $D_0$ . Hence we rewrite  $Q$  to a normal SPC query  $Q'$  defined on  $D_0$ , and let  $ans_0 = Q'(D_0)$ . Query  $Q'$  returns tuples with local attributes  $L_1$  and moreover, the set  $\text{attr}(R)$  of all attributes in each relation  $R$  that appear in  $Q$ . As will be seen shortly, we need the additional attributes to decide whether a tuple matches  $X_p$  patterns of CFKs, for horizontal expansion.

The set  $ans_0$  is generated by Algorithm Normalize, shown in Fig. 3. In addition to  $ans_0$ , Normalize also creates an initial renaming table  $M_0$ . We generate a renaming table  $M_i$  at each peer  $P_i$  visited, which keeps track of the keys of relations at the local peer. As will be seen in Section 7, the keys help us merge tuples. Table  $M_i$  consists of entries of the form  $(R_1(X) \mapsto R(Z))$ , where  $Z$  is the primary key of a relation  $R$  at the local peer  $P_0$ , and  $X$  is the primary key of  $R_1$  at  $P_i$ . Normalize produces the initial table  $M_0$ , which maps the primary key  $Z$  of each relation  $R$  to itself.

After  $ans_0$  and  $M_0$  are generated,  $P_0$  forwards them to its neighboring peers, along with the SPC\* query  $Q$ .

## 5.3. Horizontal expansion

Using CFKs  $\Sigma_{(j,i)}$  from  $P_j$  to  $P_i$ , one can extend tuples in  $ans_j$  by including relevant attributes found at  $P_i$ . Indeed, if a tuple  $t$  at  $P_j$  identifies a tuple  $t'$  at  $P_i$  via a CFK in  $\Sigma_{(j,i)}$ , then the attributes of  $t'$  are also properties of  $t$ . Hence we can extend  $t$  by adding those attributes of  $t'$  not found in  $t$ .

**Example 5.2.** Consider tuple  $t_2$  of Fig. 1(a) at peer  $P_0$  and CFK  $\varphi_1$  of Example 4.2 from  $P_0$  to  $P_2$ . Note that  $t_2$  matches the  $X_p$  pattern of  $\varphi_1$ , i.e.,  $t_2[\text{review}(\text{rating})] = 4$ . That is,  $t_2$  references a sale tuple at  $P_2$ . Indeed,  $t_5$  of Fig. 1(c) is the tuple:  $t_5[\text{review}(\text{album}, \text{artist})] = t_5[\text{sale}(\text{album}, \text{artist})]$ . Thus we can expand  $t_2$  by adding sale attributes price, label and rating, carrying the corresponding values of  $t_5$ .  $\square$

This motivates us to develop an algorithm for horizontal expansion, referred to as HExpansion and shown in Fig. 4. The algorithm takes as input  $ans_j$ ,  $\Sigma_{(j,i)}$  and a renaming table  $M_j$  forwarded from  $P_j$ . It returns as output a set  $\mathcal{Q}_{(i,h)}$  of queries for computing  $ans_i$ , and a revised renaming table  $M_i$  at  $P_i$ .

### Procedure HExpansion

*Input:* An answer set  $ans_j$ , a set  $\Sigma_{(j,i)}$  of CFKs, and a renaming table  $M_j$ .

*Output:* A set  $\mathcal{Q}_{(i,h)}$  of queries, and a renaming table  $M_i$ .

1.  $\mathcal{Q}_{(i,h)} := \emptyset$ ;  $M_i := \emptyset$ ;
2. **for** each CFK  $\varphi = (R_1[X] \subseteq R_2[Y], t_p[X_p, Y_p]) \in \Sigma_{(j,i)}$  **do**
3.     **if**  $(R_1(X) \mapsto R(Z)) \in M_j$  **then**
4.          $Q_\varphi := "(\sigma_{X_p=t_p[X_p]} ans_j) \bowtie_{R_1[X]=R_2[Y]}^l R_2"$ ;
5.          $\mathcal{Q}_{(i,h)} := \mathcal{Q}_h \uplus Q_\varphi$ ;     /\* outer union \*/
6.          $M_i := M_i \cup \{R_2(Y) \mapsto R(Z)\}$ ;
7. **return** ( $\mathcal{Q}_{(i,h)}, M_i$ );

Fig. 4. Algorithm HExpansion.

**Queries.** For each  $\varphi = (R_1[X] \subseteq R_2[Y], t_p[X_p, Y_p])$  in  $\Sigma_{(j,i)}$ , a query  $Q_\varphi$  is generated (line 4 of Fig. 4):

$$Q_\varphi = (\sigma_{X_p = t_p[X_p]} \text{ans}_j) \bowtie_{R_1[X] = R_2[Y]}^l R_2,$$

where  $\bowtie_C^l$  denotes left outer join with condition  $C$ .

Query  $Q_\varphi$  first selects those tuples  $t$  in  $\text{ans}_j$  such that  $t[X_p] = t_p[X_p]$ , and then identifies  $R_2$  tuple  $t'$  in  $D_i$  such that  $t[X] = t'[Y]$ . It expands  $t$  by adding  $t[B] = t'[B]$  for all attributes  $B$  that are not yet in  $t$ . Observe that such  $t'$  is *unique* if it exists, since  $Y$  is the primary key of  $R_2$ . As mentioned earlier,  $t$  carries additional attributes found at peer  $P_j$  in order to determine whether it matches the  $X_p$  pattern of  $\varphi$ .

**Example 5.3.** For the CFK  $\varphi_1$  given in Example 4.2, Algorithm HExpansion generates query  $Q_{\varphi_1}$  as follows:

```
select album, artist, price, label, rating
from ans0 t LEFT OUTER JOIN sale s on
t[rating] = 4 and t.album = s.album and t.artist = s.artist
```

Similarly for  $\varphi_2$  of Example 4.2,  $Q_{\varphi_2}$  is generated.  $\square$

**Answer sets.** Define  $Q_{(i,h)}$  to be the outer union of all the queries in  $Q_{(i,h)}$ , i.e.,  $Q_{(i,h)} = \sqcup_{\varphi \in \Sigma_{(j,i)}} Q_\varphi$ . When executed on database  $D_i$  at  $P_i$ ,  $Q_{(i,h)}$  produces a set  $Q_{(i,h)}(D_i)$ . Using table  $M_j$ , we rename attributes of the tuples in  $Q_{(i,h)}(D_i)$  to generate two sets of tuples:  $\text{new}_i$  to be sent back to the local peer  $P_0$ , and  $\text{ans}_i$  to be forwarded to  $P_i$ 's neighbors.

The set  $\text{new}_i$  includes those tuples in  $Q_{(i,h)}(D_i)$  with newly added non-null attributes. Further, we restore the names of the local attributes in  $L_1$  by applying  $M_i$  to  $\text{new}_i$ , in order to facilitate tuple merging.

To generate  $\text{ans}_i$ , we need to rename the attributes of the tuples in  $Q_{(i,h)}(D_i)$ . For each tuple  $t$  in  $Q_{(i,h)}(D_i)$ , we rename its attributes by substituting  $R_2[Y]$  for  $R_1[X]$  if  $t$  is generated by  $Q_\varphi(D_i)$ , with the CFK  $\varphi = (R_1[X] \subseteq R_2[Y], t_p[X_p, Y_p])$ . The set  $\text{ans}_i$  includes the renamed tuples. We also create renaming table  $M_i$  at peer  $P_i$  (line 6 of Fig. 4).

#### 5.4. Vertical expansion

Suppose that  $Q_j$  is a rewriting of  $Q$  on  $D_j$ . Based on  $\Sigma_{(j,i)}$ , we can further rewrite  $Q_j$  into an  $\text{SPC}^*$  query  $Q_i$  that can be normalized as an  $\text{SPC}$  query defined on the database  $D_i$  at

peer  $P_i$ . When the  $\text{SPC}$  query is evaluated on  $D_i$ , it may find new tuples missing from the local peer  $P_0$ .

**Example 5.4.** Recall query  $Q_0$  of Example 1.1, posed on peer  $P_0$  of Fig. 1(b). Using CFKs  $\varphi_1$  and  $\varphi_2$  of Example 4.2 from  $P_0$  to  $P_2$ , one can rewrite  $Q_0$  into  $\text{SPC}^*$  query  $Q_2$ :

```
select album; price, label; release
from sale
where artist="Denver, J" and rating="high"
```

Query  $Q_2$  can be rewritten into  $\text{SPC}$  query  $Q_2'$  that is defined on the sale relation, using a variation of Algorithm Normalize of Fig. 3. When  $Q_2'$  is evaluated on the sale data of Fig. 1(c) at  $P_2$ , it returns  $t_4$ , a tuple not found by  $Q_0$  at  $P_0$ .  $\square$

Indeed, CFKs specify contextual schema matching of [30], which is an extension of conventional schema matching. Below we outline an algorithm for query rewriting based on CFKs, extending the method of [30].

The algorithm, denoted as VExpansion, is given in Fig. 5. Suppose that  $Q_j = \Pi_L(Q_c)$ , where  $Q_c = \sigma_F(E_c)$  and  $E_c = R_1 \times \dots \times R_l$ . We ignore MK here since the matching keys are only needed for tuple merging, the final step to be done at the local peer. The rewriting consists of two steps.

(1) *Relation atoms.* For each relation atom  $R$  in  $E_c$  and each CFK  $\phi = (R[X] \subseteq R'[Y], t_p[X_p, Y_p])$  from  $R$  to  $R'$  in  $\Sigma_{(j,i)}$ , we generate a query  $q_\phi = \sigma_{F_\phi} R'$ , where  $F_\phi$  involves only equality atoms defined in terms of attributes in  $R$ , replaced with their counterparts in  $R'$  (line 3 of Fig. 5).

More specifically, for each  $A = 'a'$  in  $F$  such that  $A$  is an attribute of  $R$ , if  $A \in X$  and  $A$  corresponds to  $B \in Y$  via  $\phi$ , then we substitute  $B$  for  $A$ . If  $A \notin X$  but  $t_p[X_p]$  entails  $A = 'a'$ , then we replace  $A = 'a'$  with  $Y_p = t_p[Y_p]$ . Query  $q_\phi$  is well defined if  $\sigma_{F_\phi}$  does not contain any attributes of  $R$  after the substitutions. Similarly  $A = B$  is rewritten if both  $A$  and  $B$  are  $R$  attributes.

**Example 5.5.** Query  $Q_2$  of Example 5.4 is  $q_{\varphi_1}$ . In particular,  $\text{rating} = "4"$  is replaced with  $\text{rating} = "high"$  since the  $X_p$  pattern of  $\varphi_1$  entails  $\text{rating} = "4"$ , and its  $Y_p$  pattern is  $\text{rating} = "high"$ . In contrast,  $q_{\varphi_2}$  is not well defined, since the  $X_p$  pattern of  $\varphi_2$  does not entail  $\text{rating} = "4"$  and the review attribute  $\text{rating}$  cannot be removed from  $Q_{\varphi_2}$ .  $\square$

---

#### Procedure VExpansion

*Input:* Query  $Q_j = \Pi_L(\sigma_F(R_1 \times \dots \times R_l))$ , and a set  $\Sigma_{(j,i)}$  of CFKs.

*Output:* A rewriting  $Q_i$  of  $Q_j$  defined on  $D_i$ .

1. **for** each  $R$  in  $(R_1, \dots, R_l)$  **do**
  2.      $F_R := \text{true};$
  3.     **for** each CFK  $\phi = (R[X] \subseteq R'[Y], t_p[X_p, Y_p]) \in \Sigma_{(j,i)}$  **do**
  4.         **if**  $F_\phi$  is defined **then**  $F_R := "F_R \vee F_\phi";$
  5.      $Q_R := "\sigma_{F_R} R";$
  6.      $Q_i := "\Pi_L(\rho(\sigma_{F_i}(Q_{R_1} \times \dots \times Q_{R_l})))";$
  7.     **if**  $Q_i$  is well-defined **then return**  $Q_i;$
  8.     **else return** "Undefined";
- 

**Fig. 5.** Algorithm VExpansion.

Recall that for each relation  $R$  in  $S_j$ , there exists at most one relation  $R'$  in  $S_i$  such that  $\Sigma_{(j,i)}$  contains CFKs from  $R$  to  $R'$ . Let  $\Sigma_R$  denote the set of CFKs from  $R$  to  $R'$  in  $\Sigma_{(j,i)}$ . Define query  $Q_R = \sigma_{F_R} R'$ , where  $F_R = \bigvee_{\phi \in \Sigma_R} F_\phi$  (line 5 of Fig. 5) which is equivalent to the union of well-defined  $q_\phi$ 's.

(2) Query  $Q_i$ . We write  $Q_i$  as  $\Pi_L(\rho(\sigma_{F_i}(Q_{R_1} \times \dots \times Q_{R_l})))$ , where  $F_i$  consists of equality of the form  $RA = R'B$  for distinct  $R$  and  $R'$ , renamed with attributes in  $D_i$ . Attributes corresponding to the primary keys of the local peer are renamed by  $\rho$  (derived from renaming table  $M_i$ ), along the same lines as described above. Query  $Q_i$  is well defined if so is every  $Q_{R_i}$ , and if  $F_i$  does not contain attributes in  $D_j$ .

**Example 5.6.** Given  $Q_0$  and  $\varphi_1, \varphi_2$  described in Example 5.4, Algorithm VExpansion returns the query  $Q_2$  given in Example 5.4.

In contrast, given  $Q_2$  and CFKs  $\varphi_3, \varphi_4$  of Example 4.2 from  $P_2$  to  $P_3$ , VExpansion does not return any well-defined query. Indeed,  $Q_2$  cannot be rewritten to a query that is defined on the CD database at peer  $P_3$ , since the attribute rating does not find a counterpart in CD, and hence, equality atom rating="4" in the where clause of  $Q_2$  cannot be translated.  $\square$

Observe the following. (a) There is a simple procedure to check whether  $Q_i$  is well defined, in quadratic time in the sizes of the query  $Q_i$  and the schema  $S_i$  (details omitted). (b) Query  $Q_i$  can be readily converted to an  $\text{SPC}^*$  query of the form given in Section 3, in linear-time in the size of  $Q_i$ . Indeed, for each  $i \in [1, l]$ , suppose that  $R_i$  in  $S_j$  is mapped to  $R'_i$  in  $S_i$  via the CFKs. Then  $Q_i$  can be rewritten as  $\Pi_L(\rho(\sigma_{F'_i}(R'_1 \times \dots \times R'_l)))$ , where  $F'_i$  is the conjunction of  $F_i$  and  $F_{R'_i}$  for all  $i \in [1, l]$ . We refer to this  $\text{SPC}^*$  query also as  $Q_i$ .

**Answer sets.** When  $Q_i$  is well-defined, it can be rewritten to an  $\text{SPC}$  query  $Q'_i$  defined on  $D_i$ , along the same lines as Algorithm Normalize. The tuples returned by  $Q'_i(D_i)$  are added to  $\text{ans}_i$  and  $\text{new}_i$ . As shown in Fig. 2, peer  $P_i$  sends  $\text{new}_i$  back to the local peer  $P_0$ ; it forwards the  $\text{SPC}^*$  query  $Q_i$ , table  $M_i$  and the answer set  $\text{ans}_i$  to its neighbors.

### 5.5. Optimization

To simplify the discussion we have so far included entire tuples in  $\text{ans}_i$  and  $\text{new}_i$ . This is often unnecessary. Below we present optimization methods to reduce the communication cost by removing redundancies from  $\text{new}_i$  and  $\text{ans}_i$ .

(1) *Reducing new<sub>i</sub>*. Whenever a new tuple  $t$  is found at  $P_i$ , we generate a unique  $\text{id}(t)$  (by associating the  $\text{id}$  of  $P_i$  with it), which is sent to the local peer  $P_0$  and neighbors of  $P_i$ . When new attributes are found for  $t$  later at some peer  $P_r$ , we do not include the entire tuple in  $\text{new}_i$ ; instead, it suffices to send only  $\text{id}(t)$  and the new attributes to  $P_0$ .

(2) *Reducing ans<sub>i</sub>*. This set is needed for horizontal expansion at neighboring peers  $P_r$ , via queries  $Q_{(i,r)}$ . As will be seen shortly,  $Q_{(i,r)}$  can be generated at peer  $P_i$ . Recall from Fig. 4 that each query in  $Q_{(i,r)}$  is of the form  $(\sigma_{X_p = t_p[X_p]} \text{ans}_i) \bowtie_{R_1[X] = R_2[Y]}^L R_2$ . The condition  $X_p = t_p[X_p]$  can be checked earlier at  $P_i$ . Hence, we only need to send to  $P_r$  a subset  $T_\phi$  of  $\text{ans}_i$ , consisting of those tuples of  $\text{ans}_i$  that

satisfy this condition, i.e., those tuples of  $\text{ans}_i$  that can be possibly expanded at  $P_r$  instead of  $\text{ans}_i$ . Better still, for each  $t$  of these tuples, we only send  $\text{id}(t)$  and attributes  $t[X]$ , rather than the entire tuple. Accordingly the query above can be simplified to  $T_\phi \bowtie_{R_1[X] = R_2[Y]}^L R_2$ .

## 6. A top-K algorithm for evaluating polymorphic queries

The conceptual evaluation strategy given in Section 5 is based on search by flooding: each peer  $P_i$  forwards its answer set  $\text{ans}_i$  to all of its neighboring peers. When  $\text{ans}_i$  is large, it may incur high communication cost. In practice, however, one often wants only top  $K$  tuples in the answer [17].

In light of these, we next develop a top-K algorithm for evaluating polymorphic queries. For predefined numbers  $K$  and  $m$ , and given an  $\text{SPC}^*$  query  $Q$ , the algorithm evaluates  $Q$  in a P2P system  $\mathcal{P}$  and returns  $K$  high-quality tuples as the answer to  $Q$ . Instead of search by flooding, each peer  $P_i$  sends at most  $K$  tuples to at most  $m$  neighboring peers, selected based on a quality model. As will be verified by our experimental results, this algorithm significantly reduces the communication cost and is still able to find quality answers.

Below we present the quality model, the strategy for selecting peers and tuples, and the top-K algorithm.

### 6.1. A quality model for search

Consider a peer  $P_i$  at which a set  $\text{ans}_i$  is already computed. Let  $\mathcal{P}_i$  denote the set of neighboring peers of  $P_i$ . We want to (a) select a set  $\mathcal{P}_i^s$  of at most  $m$  peers from  $\mathcal{P}_i$ , and (b) for each peer  $P_r \in \mathcal{P}_i^s$ , choose a set  $\text{ans}_{(i,r)}$  of at most  $K$  tuples from  $\text{ans}_i$  to forward to  $P_r$ , such that these peers maximally expand the chosen tuples horizontally and vertically.

To determine  $\mathcal{P}_i^s$  and  $\text{ans}_{(i,r)}$ , we introduce a quality model, based on certain statistics of neighboring peers.

**Statistics.** Consider the P2P system  $\mathcal{P}$  described in Section 5. For each neighbor  $P_r$  of  $P_i$ , we assume that the following information about  $P_r$  is stored at  $P_i$ : (a) the schema  $S_r$  of the database  $D_r$  at  $P_r$ , (b) the set  $\Sigma_{(i,r)}$  of CFKs from  $P_i$  to  $P_r$ , and (c) statistics about  $D_r$ : for each attribute  $B$ , the cardinality  $|\text{adom}(B)|$  of its active domain in relation  $R_B$  where  $B$  appears, and the cardinality  $|R_B|$  of  $R_B$ ; we estimate the selectivity  $B\%$  of  $B$  by  $|\text{adom}(B)|/|R_B|$ .

Given these, at peer  $P_i$  we derive query  $Q_r$  for vertical expansions at peer  $P_r$ , by algorithm VExpansion of Fig. 5. Let the  $\text{SPC}^*$  query  $Q_i$  be  $\Pi_L(Q_c)$ , where  $L = (L_1; L_2; \alpha)$ .

**Quality model.** We denote by  $\text{score}(r)$  the amount of new information (attributes and tuples) that  $P_r$  may add to  $\text{ans}_i$ :

$$\text{score}(r) = \text{score}(r, h) + \text{score}(r, v),$$

where (a)  $\text{score}(r, h)$  assesses explicit attributes in  $L_2$  and implicit attributes for  $\alpha$  (if specified) added by horizontal expansion queries  $Q_{(r,h)}$ , associated with weights  $w_e$  and  $w_\alpha$ , respectively; and (b)  $\text{score}(r, v)$  estimates new tuples found by vertical expansion query  $Q_r$ , with weight  $w_v$ .

(a) We calculate  $\text{score}(r, h)$  using  $\text{ans}_i$  and  $\Sigma_{(i, r)}$ . For each tuple  $t \in \text{ans}_i$  and each CFK  $\varphi = (R_1[X] \subseteq R_2[Y], t_p[X_p, Y_p])$  in  $\Sigma_{(i, r)}$ , we define  $\text{score}(t, \varphi) = w_e * n_e + w_\alpha * n_\alpha$  if  $t[X_p] = t_p[X_p]$ , and let  $\text{score}(t, \varphi)$  be 0 otherwise. Here  $n_e$  is the number of  $R_2$  attributes in  $L_2$  of  $Q_i$ , and  $n_\alpha$  is the number of  $R_2$  attributes in neither  $L_1$  nor  $L_2$ . Since  $Y$  is a key of  $R_2$ , if  $t[X_p] = t_p[X_p]$ , there exists at most one  $R_2$  tuple  $t'$  referenced by  $t$ .

We define  $\text{score}(t) = \text{Sum}_{\varphi \in \Sigma_{(i, r)}} \text{score}(t, \varphi)$ , the sum of  $\text{score}(t, \varphi)$  when  $\varphi$  ranges over all CFKs in  $\Sigma_{(i, r)}$ .

We pick  $K$  tuples  $t$  from  $\text{ans}_i$  with the highest nonzero  $\text{score}(t)$ , and let  $\text{ans}_{(i, r)}$  be the set of these tuples. We define

$$\text{score}(r, h) = \text{Sum}_{t \in \text{ans}_{(i, r)}} \text{score}(t).$$

(b) We compute  $\text{score}(r, v)$  based on vertical expansion query  $Q_r$  and the statistics about  $D_r$ . Recall from Fig. 5 that  $Q_r = \Pi_L(\rho(\sigma_{F_r}(Q_{R_1} \times \dots \times Q_{R_l})))$ , where each  $Q_{R_i}$  is an SPC query  $\sigma_{F_{R_i}}(R_s')$ , derived from CFKs in  $\Sigma_{(i, r)}$ .

We define  $\text{score}(Q_{R_i})$  and  $\text{score}(r, v)$  as follows:

$$\text{score}(Q_{R_i}) = (w_e * n_e + w_\alpha * n_\alpha) * |R_s'| * \text{Product}_{B \in F_{R_i}} B\%,$$

$$\text{score}(r, v) = w_v * \text{Product}_{C \in F_r} C\% * \text{Product}_{s \in [1, l]} \text{score}(Q_{R_s}).$$

Here  $n_e$  and  $n_\alpha$  are the numbers of new attributes in  $R_s'$  as described above,  $B$  ranges over attributes in condition  $F_{R_i}$ , and  $\text{Product}_{B \in F_{R_i}} B\%$  estimates the probability that  $F_{R_i}$  is satisfied by the multiplication of the selectivity of the attributes in  $F_{R_i}$ . Intuitively,  $\text{score}(r, v)$  estimates the number of tuples returned by  $Q_r$ , assessed by attributes added.

The larger  $\text{score}(r)$  is, the more new information  $P_r$  may contribute and thus the higher its ranking is.

**Example 6.1.** Referring to Fig. 1(b), let us consider peer selection at  $P_2$ . As shown in Examples 5.2 and 5.4, the answer set at  $P_2$  consists of  $s_1$  (a combination of  $t_2$  and  $t_5$ ) and  $s_2$  (i.e.,  $t_4$ ). Assume that  $P_3$  and  $P_7$  allow horizontal expansion only, vertical only at  $P_5$ , and horizontal and vertical expansions at  $P_4$ . Let  $w_e = 0.7$ ,  $w_\alpha = 0.2$ , and  $w_v = 0.1$ .

By checking the CFKs, we get  $(s_2, 1, 1, P_3)$ , indicating that  $P_3$  can expand  $s_2$  with  $n_e = 1$  (attribute release) and  $n_\alpha = 1$  (rank). Similarly for  $(s_1, 1, 0, P_4)$ ,  $(s_1, 1, 0, P_7)$ ,  $(s_1, 0, 0, P_3)$ ,  $(s_2, 0, 0, P_4)$ , and  $(s_2, 0, 0, P_7)$ . Then at  $P_3$ ,  $\text{score}(s_2) = (0.7 * 1 + 0.2 * 1) = 0.9$  and  $\text{score}(s_1) = 0$ . Similarly we get scores for  $s_1$  and  $s_2$  at  $P_4$  and  $P_7$ . Then for horizontal expansion,  $\text{score}(P_3, h) = 0.9$ ,  $\text{score}(P_4, h) = 0.7$ , and  $\text{score}(P_7, h) = 0.7$ . Peer  $P_2$  next calculates the scores for vertical expansion. By checking the statistics of the data at its neighbors,  $P_2$  gets  $(P_4, 100, 1, 1, 0.01, 0.2)$ , indicating that  $P_4$  has a relation of 100 tuples with  $n_e = 1$ ,  $n_\alpha = 1$ , 1% of the tuples satisfying artist="Denver, J", and 20% with a rate of "high". Similarly we get  $(P_5, 200, 0, 2, 0.02, 0.3)$ . The scores are  $\text{score}(P_3, v) = 0$ ,  $\text{score}(P_4, v) = 0.1 * 100 * (0.7 * 1 + 0.2 * 1) * 1\% * 20\% = 0.018$ ,  $\text{score}(P_5, v) = 0.048$ , and  $\text{score}(P_7, v) = 0$ .

Taking these together, we have that  $\text{score}(P_3) = \text{score}(P_3, h) + \text{score}(P_3, v) = 0.9$ ,  $\text{score}(P_4) = 0.718$ ,  $\text{score}(P_5) = 0.048$  and  $\text{score}(P_7) = 0.7$ . □

To simplify the discussion we assume minimal statistics about neighboring peers. The quality model and algorithms can be readily improved if  $P_i$  collects statistics about peers that can be reached within  $h$  hops, for a constant  $h > 1$ .

**Peer and tuple selection.** Based on the quality model, we develop an algorithm for selecting a set  $\mathcal{P}_i^s$  of  $m$  peers and for each  $P_r \in \mathcal{P}_i^s$ , a set  $\text{ans}_{(i, r)}$  of  $K$  tuples. The algorithm, denoted by SelectPeers, is shown in Fig. 6. For each

---

#### Procedure SelectPeers

Input:  $K, m, \text{ans}_i, \mathcal{P}_i$  and for each neighbor  $P_r$ ,  $\Sigma_{(i, r)}$ ,  $Q_r$  and  $\mathcal{Q}_{(r, h)}$ .

Output:  $m$  peers  $\mathcal{P}_i^s$  and for each  $P_r \in \mathcal{P}_i^s$ ,  $K$  tuples  $\text{ans}_{(i, r)}$ .

1. for each peer  $P_r \in \mathcal{P}_i$  do
  2.   for each  $t \in \text{ans}_i$  do   /\* for horizontal expansion \*/
  3.     for each  $\varphi \in \Sigma_{(i, r)}$  do
  4.       compute  $\text{score}(t, \varphi)$ ;
  5.        $\text{score}(t) := \text{Sum}_{\varphi \in \Sigma_{(i, r)}} \text{score}(t, \varphi)$ ;
  6.      $\text{ans}_{(i, r)} :=$  the set of  $K$  tuples  $t$  in  $\text{ans}_i$  with the highest nonzero  $\text{score}(t)$ ;
  7.      $\text{score}(r, h) := \text{Sum}_{t \in \text{ans}_{(i, r)}} \text{score}(t)$ ;
  8.   /\* for vertical expansion \*/
  9.   for each  $Q_{R_i}$  in  $Q_r$  do
  10.     /\*  $Q_r = \Pi_L(\rho(\sigma_{F_r}(Q_{R_1} \times \dots \times Q_{R_l})))$  \*/
  11.      $\text{score}(Q_{R_i}) := (w_e * n_e + w_\alpha * n_\alpha) * |R_s'| * \text{Product}_{B \in F_{R_i}} B\%$ ;
  12.      $\text{score}(r, v) := w_v * \text{Product}_{B \in F_r} B\% * \text{Product}_{s \in [1, l]} \text{score}(Q_{R_s})$ ;
  13.      $\text{score}(r) := \text{score}(r, h) + \text{score}(r, v)$ ;
  14.      $\mathcal{P}_i^s :=$  the set of  $m$  peers  $P_r$  with the highest nonzero  $\text{score}(r)$ ;
  15. return  $\mathcal{P}_i^s$  and the set  $\text{ans}_{(i, r)}$  for each  $P_r \in \text{score}(r)$ ;
- 

Fig. 6. Algorithm SelectPeers.

neighbor  $P_r$  of  $P_i$ , it computes  $ans_{(i,r)}, score(r,h)$  (lines 2–7),  $score(r,v)$  (lines 8–10) and  $score(r)$  (line 11) as described above. It then selects  $m$  peers with the highest nonzero scores, and builds the set  $\mathcal{P}_i^s$  with these peers (line 12). It returns  $\mathcal{P}_i^s$  and a set  $ans_{(i,r)}$  for each  $P_r$  in  $\mathcal{P}_i^s$  (line 13).

**Example 6.2.** When  $m=2$  and  $K=2$ , SelectPeers selects  $P_3$  and  $P_4$  at peer  $P_2$  based on the scores computed in Example 6.1. The sets of tuples to be sent to its neighbors  $P_3$  and  $P_4$  are  $\{s_2\}$  and  $\{s_1\}$ , respectively.  $\square$

## 6.2. A top-K algorithm

We next present the top-K algorithm, which revises the conceptual evaluation strategy by leveraging the quality model and Procedure SelectPeers.

**Initial answer.** Upon receiving  $Q$  and a predefined TTL, the local peer  $P_0$  does the following. (1) It generates  $ans_0$  by using Algorithm Normalize of Fig. 3. (2) For each neighbor  $P_r$ , it generates queries  $Q_{(r,h)}$  and  $Q_r$  for horizontal and vertical expansions, by HExpansion and VExpansion of Figs. 4 and 5, respectively. (3) By Algorithm SelectPeers, it selects at most  $m$  neighbors and forwards at most  $K$  tuples to each of these peers. Also forwarded are queries  $Q_{(r,h)}$  and  $Q_r$ , and renaming table  $M_0$ . It also decreases TTL by 1 and forwards it to the selected neighbors.

In contrast to the conceptual strategy, the queries for expansions at the neighbors of  $P_0$  are generated at  $P_0$ . Further, at most  $K$  tuples are forwarded to at most  $m$  selected peers, instead of search by flooding.

**Expand and forward.** When peer  $P_i$  receives tuples  $ans_j$  and queries  $Q_{(i,h)}$  and  $Q_i$  from  $P_j$ , it does the following.

1. It executes  $Q_{(i,h)}$  and  $Q_i$  against its local database  $D_i$ , to expand  $ans_j$  horizontally and vertically, as described in its counterpart of the conceptual strategy.
2. It generates sets  $new_i$  and  $ans_i$  of tuples, and sends  $new_i$  back to the local peer  $P_0$ .
3. If  $TTL=0$ , no more expansion is conducted. Otherwise it produces expansion queries for its neighbors, selects at most  $m$  peers, forwards the queries and at most  $K$  tuples to these neighbors along with renaming table  $M_i$ , as described in steps (2) and (3) of the initial answer stage. TTL is decreased by 1 and is also forwarded to its selected neighbors.

As described in Section 5.5, only necessary parts of  $new_i$  and  $ans_i$  are sent to  $P_0$  and selected neighbors, respectively.

**Tuple merging.** When TTL expires or when the local peer  $P_0$  receives no more new tuples or attributes for a certain period of time,  $P_0$  merges tuples and handles conflicts as will be described in Section 7. It then selects  $K$  tuples as follows. (1) For each tuple  $t$  collected by  $P_0$ , define

$$ranking(t) = w_e * n_e + w_x * n_x,$$

where  $n_e$  and  $n_x$  are the numbers of its explicit ( $L_2$ ) attributes and implicit ( $\alpha$ ) attributes with non-null values in the input query  $Q$ , respectively, and  $w_e$  and  $w_x$  are

weights described in Section 6.1. (2) Peer  $P_0$  selects  $K$  tuples with the highest ranking scores (or all the tuples if there exist less than  $K$  tuples), and returns them as the answer to query  $Q$ .

**Example 6.3.** We show how Algorithm TopKPoly evaluates the  $spc^*$  query  $Q_0$  of Example 1.1 in the P2P system of Fig. 1(b). Let  $K=2$ ,  $m=2$  and  $TTL=2$ . Upon receiving  $Q_0$ , the local peer  $P_0$  finds  $ans_0=\{t_2\}$  by Normalize. It then generates queries for horizontal and vertical expansions at its neighbors  $P_2, P_5$  and  $P_1$ , including  $Q_{\phi_1}$  (Example 5.3) and  $Q_2$  (Example 5.4) for expansions at  $P_2$ . Assume that  $P_2$  is the only peer selected by SelectPeers. Then  $P_0$  decreases TTL by 1 and sends  $(Q_{\phi_1}, Q_2, M_0, ans_0, TTL=1)$  to  $P_2$ .

When  $P_2$  receives the request from  $P_0$ , it evaluates  $Q_{\phi_1}$  and  $Q_2$  against its local database sale, and gets the set  $ans_2$ :

	album	artist	price	label	rating
$s_1$ :	Greatest Hits	Denver, J	8.36	BMG	high
$s_2$ :	Take Me Home	Denver, J	5.97	Windstar	high

where  $s_1$  is found by horizontally expanding  $t_2$  with attributes in  $t_5$  via  $Q_{\phi_1}$ , and  $s_2$  by vertical expansion via  $Q_2$ . The newly added tuples and attributes are sent back to  $P_0$  as  $new_2$ .

Peer  $P_2$  then produces expansion queries for  $P_3, P_4, P_5, P_7$ . For example, horizontal expansion query  $Q_{\phi_3}$  at  $P_3$  is

```

select title, artist, label, release
from   ans2 t LEFT OUTER JOIN CD s on
       t[label]= Windstar and t. album=s.title and t. artist=s.artist

```

As shown in Example 6.2, Algorithm SelectPeers selects  $P_3$  and  $P_4$  as the top 2 peers. Then  $P_2$  sends the corresponding expansion queries to  $P_3$  and  $P_4$ , along with the renaming table  $M_2$ . Here  $M_2=\{sale(album) \mapsto review(album), sale(artist) \mapsto review(artist)\}$ . It sends tuple  $s_2$  to  $P_3$  and  $s_1$  to  $P_4$  for expansions. It also decreases TTL by 1 and sends  $TTL=0$  to  $P_3$  and  $P_4$ .

Given  $Q_{\phi_3}$  and  $ans_2$ , peer  $P_3$  evaluates  $Q_{\phi_3}$  against its CD relation, and extends tuple  $s_2$  in  $ans_2$  by adding a *release* attribute taken from  $t_8$  (Fig. 1(d)). It sends the newly added attribute back to  $P_0$  as  $new_3$ . Similarly, expansion queries are evaluated at  $P_4$ . Note that at  $P_3$  and  $P_4$ ,  $TTL=0$  and thus no further expansion is needed.

The answer set at  $P_0$  consists of

	album	artist	price	label	release
$s_1$ :	Greatest Hits	Denver, J	8.36	BMG	null
$s_2$ :	Take Me Home	Denver, J	5.97	Windstar	03/07/2006
$s_3$ :	The Greatest Hits	Denver, J	9.99	BMG	01/07/2002

where  $s_3$  is found at  $P_4$ , by vertical expansion (the *rating* attribute is omitted). As will be seen in Section 7, the tuples are merged to produce the final answer to  $Q_0$  in the P2P system.  $\square$

**Remark.** The choice of TTL,  $K$  and  $m$  values are application dependent, adapted to strike a balance between the



completeness of query answer and the cost for computing the answer. The study of TTL in traditional P2P models is also applicable to polymorphic query processing (see, e.g., [32]). For an unstructured P2P network of size  $n$ , where each peer has at most  $p$  neighbors, TTL is usually the minimal integer  $t$  satisfying that  $1 + p + p^2 + \dots + p^t \geq n$ . In this way, each peer has a chance to be visited before TTL expires. Accordingly,  $m$  is an integer between 1 and  $p$ , and the value is decided based on individual application domains.

## 7. Merging tuples and resolving conflicts

A polymorphic query may have attributes specified with type variables, namely, explicit ( $L_2$ ) attributes and implicit ( $\alpha$ ) attributes. Moreover, attributes or tuples collected from different peers may refer to the same real-world object but may have *typing conflicts* and *data conflicts*. We now present methods for instantiating type variables, merging tuples representing the same object, and for handling conflicts.

**An example.** We illustrate our methods using a query

$$Q_0' = \Pi_{(L_1:L_2)}(\sigma_F(E_C)) \text{ group\_by } \phi,$$

where  $\Pi_{(L_1:L_2)}(\sigma_F(E_C))$  is query  $Q_0$  given in Example 3.1, which is to find the album ( $L_1$ ), and price, label, release ( $L_2$ ) of John Denver's albums that received a good rating. Here  $\phi$  is the matching key  $((\text{album}, \approx), (\text{label}, =))$  given in Example 3.2.

Query  $Q_0'$  is posed on peer  $P_0$  of the P2P system depicted in Fig. 1(b). As shown in Example 6.3, a set of tuples is found and returned to  $P_0$ , denoted by  $ans$ , including:

	album	artist	price	label	release
$s_1$ :	Greatest Hits	Denver, J	8.36	BMG	null
$s_3$ :	The Greatest Hits	Denver, J	9.99	BMG	01/07/2002

These tuples represent the same real-world object.

**Method.** Our tuple merging method consists of two steps:

- partition  $ans$  into groups, such that tuples in the same group represent the same real-world object; and
- develop a single succinct representation for each group, and instantiate type variables and null attributes.

We next present the details of these steps.

**Grouping tuples.** We partition  $ans$  such that each tuple  $s$  in  $ans$  is in a group, denoted by  $\text{eq}(s)$ . When two tuples  $s$  and  $s'$  are identified to represent the same object, we merge  $\text{eq}(s)$  and  $\text{eq}(s')$  into the same group. We identify tuples based on primary keys and matching keys, as follows.

(1) Recall from Figs. 3 and 4 that we keep track of the primary key of each relation at the local peer throughout the horizontal expansion process, by propagating renaming tables. These keys are also carried by each tuple in  $ans$ , e.g.,  $s_1$  and  $s_3$  above retain the key (album, artist) of relation review at  $P_0$ . The keys allow us to identify tuples that are horizontal expansions of the same tuple. Indeed, when two tuples  $s$  and  $s'$  in  $ans$  have the same primary

key for each of the base relations, we can merge  $\text{eq}(s)$  and  $\text{eq}(s')$ .

(2) Primary keys, however, typically do not help when matching tuples resulted from vertical expansion. For example,  $s_1$  and  $s_3$  cannot be identified by primary key since  $s_1[\text{album}] \neq s_3[\text{album}]$ . In contrast, we can identify  $s_1$  and  $s_3$  by using matching key  $\phi$  as shown in Example 3.2, and merge  $\text{eq}(s_1)$  and  $\text{eq}(s_3)$ . Now  $s_1$  and  $s_3$  are in the same group  $\text{eq}(s_1)$ . Putting these together, we use matching keys to identify tuples found by vertical expansion, and primary keys to match tuples resulted from horizontal expansion.

Tuples  $s$  and  $s'$  in the same group can be merged if for each attribute  $A$ , either  $s[A] = s'[A]$  or one of them is null. In the latter case  $s[A]$  takes the non-null value.

If after the merging process, each group has a single tuple, then there is no data conflict. However, conflicts are commonly found in practice. For instance, after the merging process,  $\text{eq}(s_1)$  remains  $\{s_1, s_3\}$  and cannot be further reduced:  $s_1$  and  $s_3$  differ in their album and price attributes.

**Representing groups.** Several methods have been studied to resolve data conflicts in data integration. A naive method is to set conflicting attributes to null. A better way is to use resolution functions such as ANY, FIRST, LAST, MIN, MAX, AVG, DISCARD (e.g., [18,33,19]). Given a list  $\text{Val}$  of values with the same data type, ANY draws a random value from  $\text{Val}$ , FIRST returns the first one, LAST returns the last one; MIN, MAX and AVG return the minimum, maximum and average values, respectively, when  $\text{Val}$  consists of numerical values; DISCARD returns null if  $\text{Val}$  contains more than one value, and it returns the only value in  $\text{Val}$  otherwise. The user may choose one of these functions.

We adopt another approach, based on OR-sets proposed in [21,22] for managing incomplete information. An OR-set is a disjunction of a set of data values. For instance,  $\text{eq}(s_1)$  is represented as OR-set:

album	artist	price	label	release
{Greatest Hits, The Greatest Hits}	Denver, J	{8.36, 9.99}	BMG	01/07/2002

indicating an album of J. Denver known as either “Greatest Hits” or “The Greatest Hits”, released on 01/07/2002 by BMG, with a price of 8.36 or 9.99.

More specifically, we cope with data conflicts by representing each group as a single OR-set. Compared to other resolution methods, OR-sets provide a succinct representation for all the relevant data, *without loss of information*.

To resolve typing conflicts, for each attribute  $A$  in  $L_2$  (or  $\alpha$ ), we cast the types of the values of the  $A$  column of  $ans$  into a uniform type  $\tau_A$  by leveraging an automatic type casting mechanism, such that the type variable for  $A$  is instantiated with  $\tau_A$ . When multiple attributes are explicitly required to share the same type variable (see Section 3), such typing constraints are enforced at this stage so that these attributes have the same type. Techniques for type inference in functional programming [13] and relational queries [24] can also be incorporated when resolving typing conflicts.

**Preparing final answer.** If  $\alpha$  is not specified in the query, we need to remove redundant attributes from the groups. For instance, for query  $Q_0'$ , we remove artist from  $eq(s_1)$ :

album	price	label	release
{Greatest Hits, The Greatest Hits}	{8.36, 9.99}	BMG	01/07/2002

We sort the groups based on the number of non-null attributes in  $L_2$  (and  $\alpha$  if it is specified), as described in Section 6.2. In an OR-set, an attribute is non-null if its value contains a value that is not null. The groups with the highest ranking scores are returned as the answer to the query.

## 8. Experimental study

In this section we present an experimental study of the following three approaches to evaluating queries in P2P systems: (a) flooding, the conceptual evaluation strategy for  $SPC^*$  queries presented in Section 5, (b) TopK, the top-K algorithm developed in Section 6, and (c) mapping, the approach based on schema mapping. We focus on the impact of the size of P2P system (the number of peers) and the sizes of databases on the quality of query answers and the communication cost of these approaches. Furthermore, we evaluate the effectiveness of the tuple merging method developed in Section 7.

### 8.1. Experimental setting

We performed the experiments on a cluster of 32 Linux machines, each with a 2.00 GHz Intel(R) Xeon(R) Dual-Core processor and 8 GB of memory. A commercial DBMS was installed on each of these machines. These machines were connected with a local area network. We implemented all the algorithms (flooding, TopK, mapping) in Java. For each of flooding and TopK, we implemented two versions: one with the optimization described in Section 5.5 by shipping partial answers to neighboring peers, and the other without.

**Data and CFKs.** We implemented a data generator to produce (logical) peers, schemas, data and CFKs. The input of the generator consists of the following: (a) #peer, the number of peers, (b) #neighbor, the average number of neighbors for each peer, (c) #col, the average arity (number of attributes) of a schema, (d) #size, the average number of tuples in an instance of the schema, and (e) #CFK, the average number of CFKs in  $\Sigma_{(j,i)}$  between neighboring peers  $P_j$  and  $P_i$ .

For each  $i \in [1, \text{\#peer}]$ , a relation schema  $R_i$  was generated at peer  $P_i$ , with #col attributes in average. For each pair of neighboring peers  $P_j$  and  $P_i$ , a set  $\Sigma_{(j,i)}$  of CFKs was produced, with  $|\Sigma_{(j,i)}| \leq \text{\#CFK}$ . Each CFK is of the form

$$(R_j(\text{key}_j) \subseteq R_i(\text{key}_i), t_p[\bar{A}, \bar{B}])$$

where  $t_p[\bar{A}] = R_j[\bar{A}] = \bar{r}$ ,  $t_p[\bar{B}] = R_i[\bar{B}] = \bar{s}$ .

Here  $\text{key}_j$  (resp.  $\text{key}_i$ ) is the primary key of  $R_j$  (resp.  $R_i$ ),  $\bar{A}$  (resp.  $\bar{B}$ ) denotes randomly selected attributes from  $R_j$  (resp.  $R_i$ ), and  $\bar{r}$  and  $\bar{s}$  are randomly generated constants.

**Queries.** We generated a number of  $SPC^*$  queries of the form  $\Pi_{(L_1, L_2, \alpha)}(\sigma_F(R_0))$  for testing, where (a)  $L_1$  consists of local attributes of  $R_0$ , (b)  $L_2$  is a set of explicit attributes not in  $R_0$ , randomly selected from the schemas at other peers, (c)  $\alpha$  is to be instantiated with a list of attributes appearing in neither  $L_1$  or  $L_2$ , and (d)  $F$  consists of equality atoms defined with the attributes in  $L_1$  and constants in their corresponding domains. These queries were converted to normal  $SPC$  queries by dropping the  $L_2$  attributes when being evaluated by mapping.

**Evaluation.** We conducted four sets of experiments. The first three sets evaluated the impact of the following factors on the performance of the three algorithms: #peer ranging from 10 to 100, #size from 5k to 50k tuples, and  $K$  from 10 to 100 for TopK. As observed in [12], efficient query processing in “PDMS with tens of peers tends to be intractable”. In light of this, we opt to evaluate the performance of  $SPC^*$  queries (rather than keyword search) on unstructured P2P networks with up to 100 nodes, and defer the experimental study on larger P2P systems to future work.

In each of these experiments, we evaluated the quality of query results and the communication cost of each of the three algorithms, measured by the following: (a) #attr, the number of relevant non-null attributes in final answer sets and in top  $K$  tuples, merged by matching keys, and (b) #bytes, the number of bytes that were transferred to answer a query.

The last set of experiments evaluated the effectiveness of our tuple merging techniques on identifying tuples in the query answers returned by flooding, TopK and mapping.

When not stated otherwise, we used #peer=30, #neighbor=4, TTL=4, #size=20k, #CFK=10, #col=10,  $K=20$ , and  $m=3$  in our experiments, where  $m$  is the number of selected neighbors in TopK. The weights for horizontal expansion and vertical expansion were set as  $w_e = 0.7$ ,  $w_\alpha = 0.2$ , and  $w_v = 0.1$  (see Section 6). Each set of experiments was run 5 times and the average is reported here.

### 8.2. Experimental results

Below we report our findings from the experiments.

**Varying P2P network size.** To evaluate the scalability and the quality of query answers of the algorithms, we varied #peer from 10 to 100. We report the number of relevant non-null attributes (#attr) in Figs. 7(a) and (b). As shown in Fig. 7(a), flooding found about 5.51 times more relevant attributes than mapping. We also compared the results of TopK with the top  $K$  tuples that ranked the highest in the results of flooding and mapping. In this setting, Fig. 7(b) shows that TopK found over 59.47% of the information returned by flooding and 216% more than that of mapping.

We report the communication cost (#bytes) in Fig. 7(c), which shows that TopK had almost constant communication cost, far smaller than those of flooding and mapping. This is because TopK trades the size of answer for efficiency. The results also tell us that both TopK and flooding scale well with #peer. Moreover, our optimization

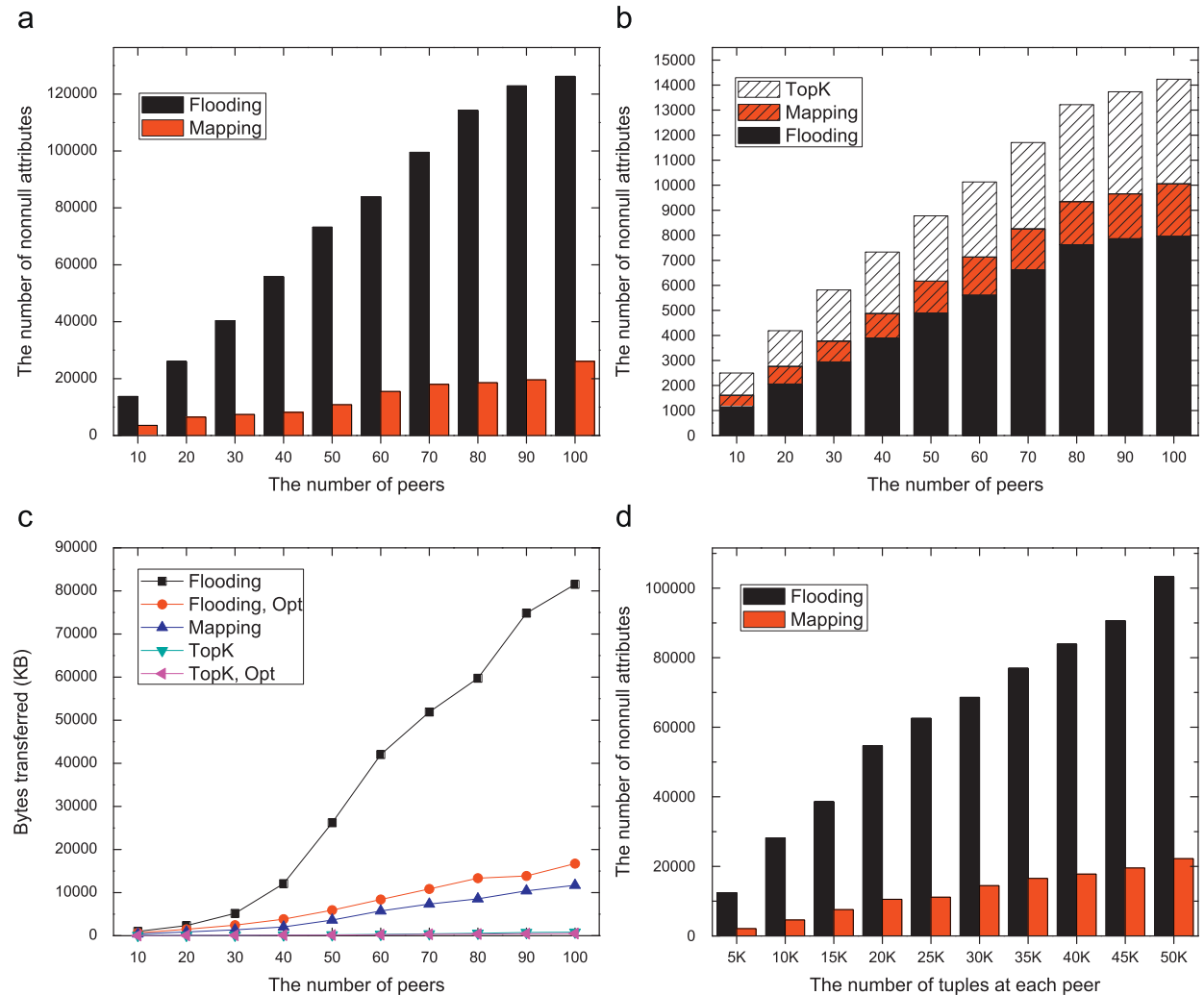
methods (Section 5.5) substantially reduces network traffic (by 92.87% for flooding and 78.71% for TopK), without hampering the quality of query answers. Among other things, with optimization the network traffic of flooding is comparable to that of mapping.

**Varying the data size.** To evaluate the impact of data size on the performance of the algorithms, we varied  $|DB|$  from 5k to 50k at each peer. The results on #attr are shown in Figs. 7(d) and (e), and on #bytes in Fig. 7(f). The results are consistent with those reported above: in average, flooding found 5.13 times more relevant non-null attributes than mapping (Fig. 7(d)); and when top  $K$  tuples are concerned, TopK found up to 84.78% of the result of flooding, and outperformed mapping by 2.53 times (Fig. 7(e)).

As shown in Fig. 7(f), the communication cost of TopK was much smaller than those of flooding and mapping, and was far less sensitive to the data size. Moreover, flooding with optimization does not incur substantial overhead

compared to mapping; this again verifies the effectiveness of the optimization techniques proposed in Section 5.5. The results also show that flooding and TopK scale well with  $|DB|$ .

**Varying  $K$ .** To evaluate the impact of  $K$  and  $m$  on the top- $K$  algorithm, we varied  $K$  from 10 to 100, while  $m$  was set to 2 or 3. The results of TopK were compared with *all the non-null attributes* found by flooding and mapping, respectively, rather than the top  $K$  tuples. As depicted in Fig. 7(g), the larger  $K$  and  $m$  were, the more relevant attributes were found by TopK, as expected. Better still, TopK outperformed mapping when  $K$  was sufficiently large, e.g., when  $K \geq 40$  and  $m = 3$  (resp.  $K \geq 60$  and  $m = 2$ ). On average, the relevant non-null attributes found by TopK were 12.94% of that of flooding, and were 1.42 times more than that of mapping. As shown in Fig. 7(h), TopK constantly incurred far less network traffic than flooding and mapping, and it scaled well with  $K$ .



**Fig. 7.** Experimental results on the quality of query answers and the communication cost. (a) #attr retrieved vs. #peer; (b) #attr (top  $K$ ) vs. #peer; (c) #bytes shipped vs. #peer; (d) #attr retrieved vs.  $|DB|$ ; (e) #attr (top  $K$ ) vs.  $|DB|$ ; (f) #bytes shipped vs.  $|DB|$ ; (g) #attr retrieved vs.  $K$ ; (h) #bytes shipped vs.  $K$ .

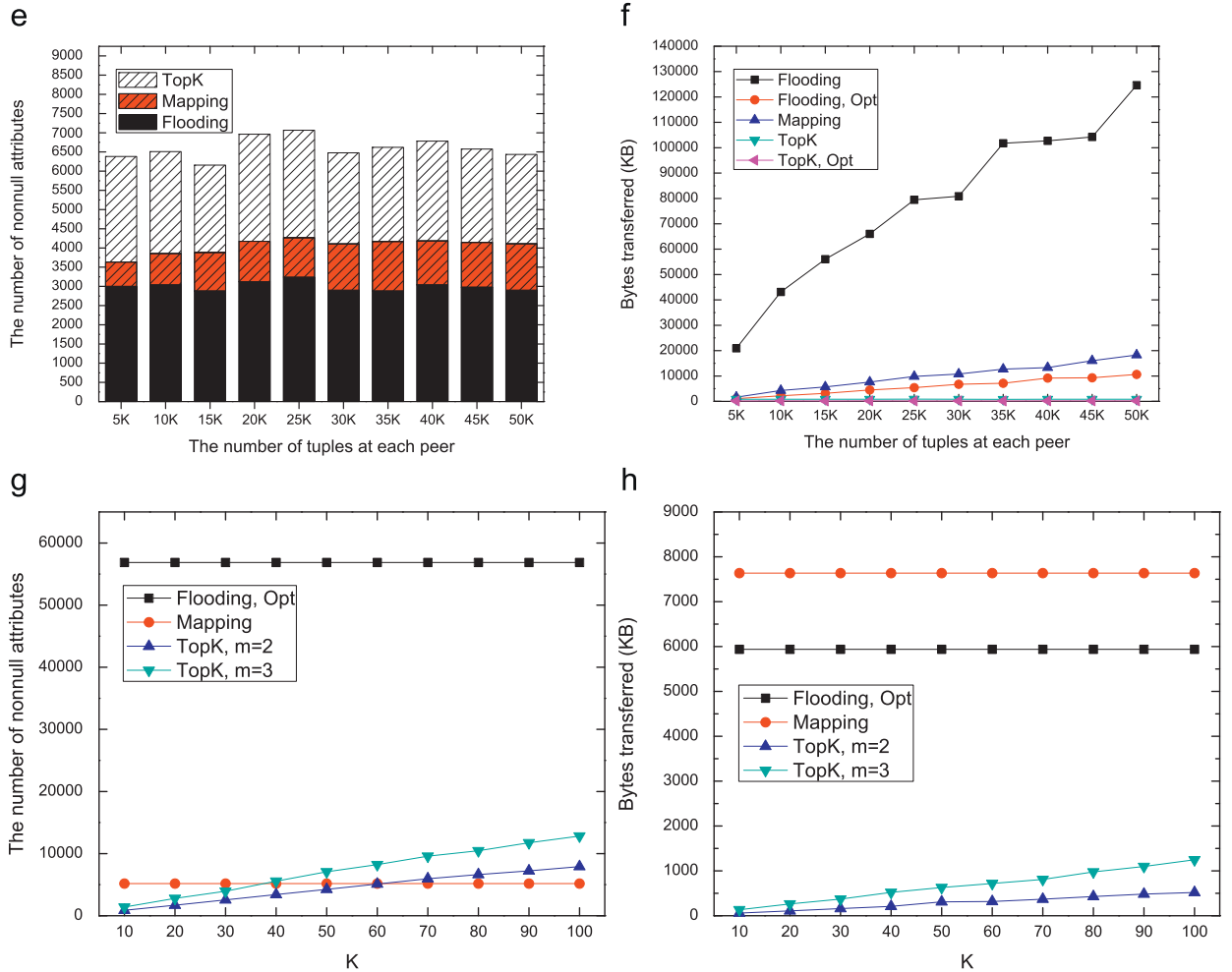


Fig. 7. (Continued)

**Tuple merging.** We also evaluated the effectiveness and scalability of our tuple merging method, varying the number of peers from 10 to 100. Fig. 8 compared the following for flooding, mapping, and TopK, respectively: (a) the number of relevant non-null attributes (#attr) in the answer sets before merging, and (b) the number of relevant non-null attributes in the sets after merging by traditional keys and by matching keys. We can see that the more peers in the network, the more #attr were merged. Fig. 8(a) shows that 71.01% of #attr in flooding were merged by traditional keys. In addition, 17.43% of these attributes were merged by matching keys (totally 88.44% of the initial answers). Fig. 8(b) tells us that on average, 17.55% of #attr in mapping were merged by traditional keys, and further, 13.19% by matching keys (i.e., 30.74% of the initial answers). Fig. 8(c) shows that #attr merged from the results of TopK was the minimum of the three, which was only 0.18% of that of flooding (resp. 6.39% against that of mapping). This is because that TopK took care to remove redundant tuples when processing the data, and thus incurred far less redundancies than its flooding and mapping counterparts.

**Summary.** From the experimental results we find the following. (a) flooding constantly outperforms the traditional mapping approach in the quality of query answers by over 5 times. In addition, when  $K$  and  $m$  are reasonably large (e.g.,  $K=40$  and  $m=3$ ), the result quality of TopK is also better than that of mapping: it finds 142% more relevant non-null attributes. These verify that polymorphic queries and their evaluation techniques are able to find more relevant information than the traditional approach. (b) Both flooding and TopK scale well with the size of P2P network and data size. In addition, TopK scales well with  $K$ . (c) TopK substantially reduces the communication cost, while it is still able to find about 84.78% of the results of flooding, and 253% more than that of mapping in average, when top  $K$  tuples that ranked the highest are concerned. (d) The optimization methods effectively reduce the communication cost, constantly by 92.87% for flooding and 78.71% for TopK. With optimization the communication cost of flooding is comparable to that of mapping. (e) The tuple merging strategy substantially improves the quality of query results, up to 83.39% for flooding and 19.86% for mapping.

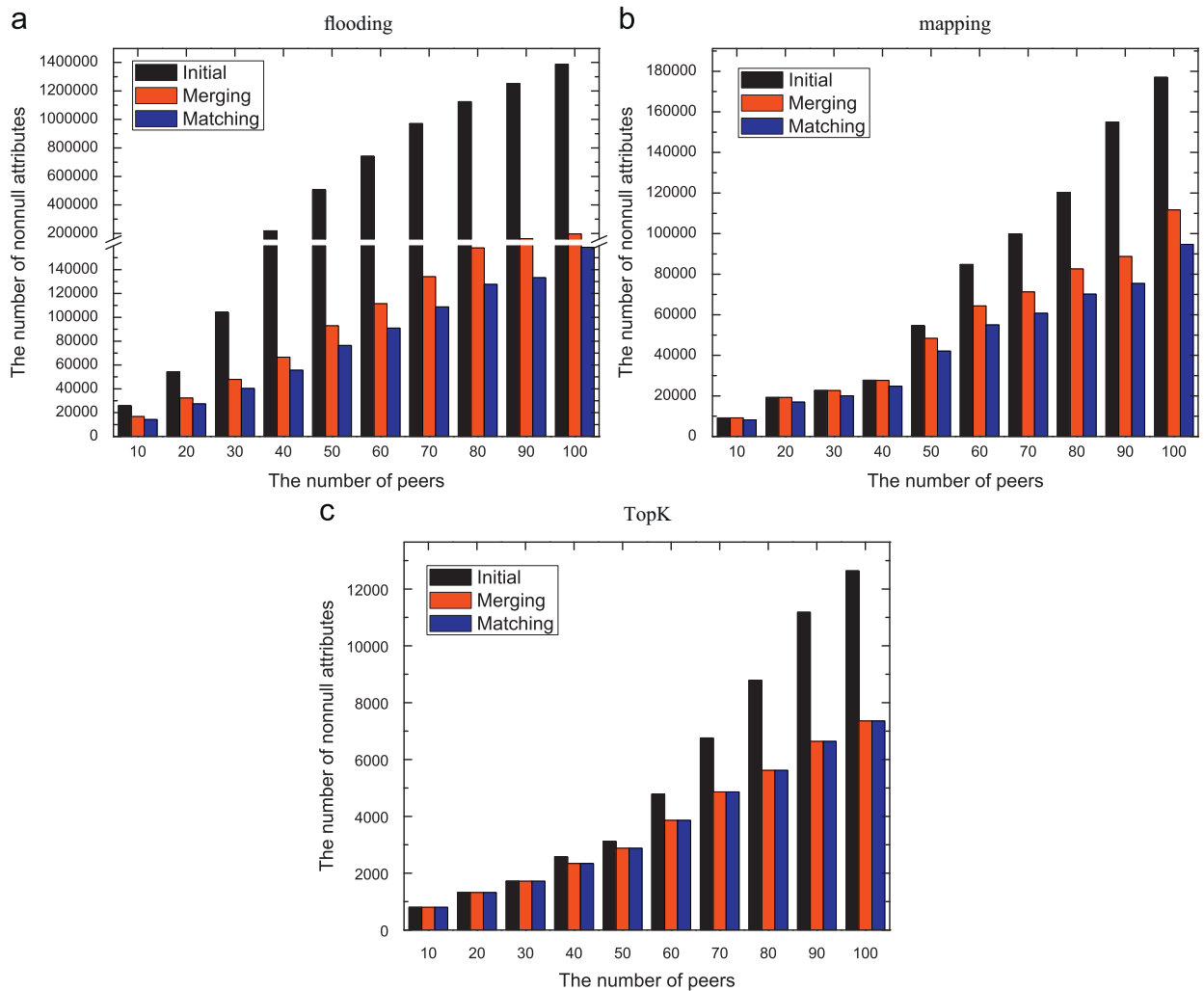


Fig. 8. The effectiveness of merging.

## 9. Conclusion

We have proposed a query model for P2P systems. Its novelty consists in the following: (a) polymorphic queries ( $SPC^*$ ) to explicitly retrieve attributes even when they are not defined at the local peer; (b) horizontal and vertical expansions, to find not only tuples but also attributes missing from the local peer; (c) CFKs to support horizontal and vertical expansions in a uniform framework; (d) a top-K algorithm for  $SPC^*$ , based on a quality model for peer and tuple selections; and (e) matching keys for identifying tuples and handling conflicts. As verified by our experimental study,  $SPC^*$  queries are able to find more relevant information than traditional  $SPC$  queries in P2P systems, without incurring high communication cost.

Several issues need further investigation. First, we are currently experimenting with larger datasets and more peer nodes. Second, we aim to extend  $SPC^*$  to polymorphic relational algebra. Third, we are exploring optimization techniques for polymorphic queries, with performance guarantees on both the efficiency and the quality of query

answers. In particular, we plan to leverage semantics indices (e.g., [34]) in order to efficiently locate peers of interest.

## Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. 60773057, 60503047).

Fan is supported in part by the RSE-NSFC Joint Project Scheme and an IBM scalable data analytics for a smarter planet innovation award.

## References

- [1] A. Bonifati, E. Chang, T. Ho, L.V. Lakshmanan, R. Pottinger, Y. Chung, Schema mapping and query translation in heterogeneous P2P XML databases, *The VLDB Journal* 19 (2) (2010) 231–256.
- [2] A.Y. Halevy, Z.G. Ives, J. Madhavan, P. Mork, D. Suciu, I. Tatarinov, The Piazza peer data management system, *TKDE* 16 (7) (2004) 787–798.
- [3] D. Calvanese, G.D. Giacomo, M. Lenzerini, R. Rosati, Logical foundations of peer-to-peer data integration, in: *Proceedings of PODS*, ACM, New York, USA, Paris, France 2004, pp. 241–251.



- [4] W.S. Ng, B.C. Ooi, K.-L. Tan, A. Zhou, PeerDB: a P2P-based system for distributed data sharing, in: Proceedings of ICDE, IEEE Computer Society, USA, Bangalore, India2003, pp. 633–644.
- [5] A. Kementsietsidis, M. Arenas, Data sharing through query translation in autonomous sources, in: Proceedings of VLDB, Morgan Kaufmann, San Francisco, USA, Toronto, Canada2004, pp. 468–479.
- [6] H.F. Witschel, T. Bohme, Evaluating profiling and query expansion methods for P2P information retrieval, in: Proceeding of P2PIR, ACM, New York, USA, Bremen, Germany2005, pp. 1–8.
- [7] A. Roth, F. Naumann, Benefit and cost of query answering in PDMS, in: Proceedings of DBISP2P, Springer, Berlin, Trondheim, Norway2005, pp. 50–61.
- [8] F.B. Kashani, C. Shahabi, Partial read from peer-to-peer databases, Computer Communications 31 (2) (2008) 332–345.
- [9] V. Kantere, D. Tsoumakos, T.K. Sellis, N. Roussopoulos, Groupeer: dynamic clustering of P2P databases, Information Systems 34 (1) (2009) 62–86.
- [10] V. Kantere, D. Tsoumakos, T.K. Sellis, A framework for semantic grouping in P2P databases, Information Systems 33 (7–8) (2008) 611–636.
- [11] K. Nakauchi, Y. Ishikawa, H. Morikawa, T. Aoyama, Peer-to-peer keyword search using keyword relationship, in: Proceedings of CCGRID03, IEEE Computer Society, USA, Tokyo, Japan2003, pp. 359–366.
- [12] K. Hose, A. Roth, A. Zeitz, K.-U. Sattler, F. Naumann, A research agenda for query processing in large-scale peer data management systems, Information Systems 33 (7–8) (2008) 597–610.
- [13] J. Garrigue, D. Rémy, Extending ML with semi-explicit higher-order polymorphism, in: Proceedings of TACS, Springer, Berlin, Sendai, Japan1997, pp. 20–46.
- [14] J. Bleiholder, F. Naumann, Data fusion, ACM Computing Surveys 41 (1) (2008) (article No. 1).
- [15] E.M. Voorhees, Query expansion using lexical-semantic relations, in: Proceedings of SIGIR, ACM/Springer, New York, USA, Dublin, Ireland1994, pp. 61–69.
- [16] M. Mitra, A. Singhal, C. Buckley, Improving automatic query expansion, in: Proceedings of SIGIR, ACM, New York, USA, Melbourne, Australia1998, pp. 206–214.
- [17] I.F. Ilyas, G. Beskales, M.A. Soliman, A survey of top-k query processing techniques in relational database systems, ACM Computing Surveys 40 (4) (2008) (article No. 11).
- [18] L.-L. Yan, M.T. Özsu, Conflict tolerant queries in aurora, in: Proceedings of CoopIS, IEEE Computer Society, USA, Edinburgh, Scotland1999, pp. 279–290.
- [19] F. Naumann, A. Bilke, J. Bleiholder, M. Weis, Data fusion in three steps: resolving schema, tuple, and value inconsistencies, IEEE Data Engineering Bulletin 29 (2) (2006) 21–31.
- [20] K.-U. Sattler, S. Conrad, G. Saake, Interactive example-driven integration and reconciliation for accessing database federations, Information Systems 28 (5) (2003) 393–414.
- [21] T. Imielinski, S.A. Naqvi, K.V. Vadaparty, Incomplete objects—a data model for design and planning applications, in: Proceedings of SIGMOD, ACM, New York, USA, Denver, Colorado1991, pp. 288–297.
- [22] T. Imielinski, S.A. Naqvi, K.V. Vadaparty, Querying design and planning databases, in: Proceedings of DOOD, Springer, Berlin, Munich, Germany1991, pp. 524–545.
- [23] A.K. Elmagarmid, P.G. Ipeirotis, V.S. Verykios, Duplicate record detection: a survey, TKDE 19 (1) (2007) 1–16.
- [24] J.V.D. Bussche, D.V. Gucht, S. Vansummeren, A crash course on database queries, in: Proceedings of PODS, ACM, New York, USA, Beijing, China2007, pp. 143–154.
- [25] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, USA, 1995.
- [26] W. Fan, X. Jia, J. Li, S. Ma, Reasoning about record matching rules, in: PVLDB, vol. 2, no. 1, 2009, pp. 407–418.
- [27] L. Bravo, W. Fan, S. Ma, Extending dependencies with conditions, in: Proceedings of VLDB, ACM, New York, USA, Vienna, Austria2007, pp. 243–254.
- [28] M. Lenzerini, Data integration: a theoretical perspective, in: PODS, 2002, pp. 61–75.
- [29] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, The VLDB Journal 10 (2001) 334–350.
- [30] P. Bohannon, E. Elnahrawy, W. Fan, M. Flaster, Putting context into schema matching, in: Proceedings of VLDB, ACM, New York, USA, Seoul, Korea2006, pp. 307–318.
- [31] P.G. Kolaitis, Schema mappings, data exchange, and metadata management, in: PODS, 2005, pp. 233–246.
- [32] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proceedings of ICS, ACM, New York, USA, New York City, US2002, pp. 84–95.
- [33] Y. Papakonstantinou, S. Abiteboul, H. Garcia-Molina, Object fusion in mediator systems, in: Proceedings of VLDB, Morgan Kaufmann, San Francisco, USA, India1996, pp. 413–424.
- [34] L. Galanis, Y. Wang, S.R. Jeffery, D.J. DeWitt, Locating data sources in large distributed systems, in: Proceedings of VLDB, Morgan Kaufmann, San Francisco, USA, Berlin, Germany2003, pp. 874–885.